A multi-objective evolutionary approach to simulation-based optimisation of real-world problems

Anna Syberfeldt

A dissertation submitted to De Montfort University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

January 2009

Research sponsored by University of Skövde

Abstract

This thesis presents a novel evolutionary optimisation algorithm that can improve the quality of solutions in simulation-based optimisation. Simulation-based optimisation is the process of finding optimal parameter settings without explicitly examining each possible configuration of settings. An optimisation algorithm generates potential configurations and sends these to the simulation, which acts as an evaluation function. The evaluation results are used to refine the optimisation such that it eventually returns a high-quality solution. The algorithm described in this thesis integrates multi-objective optimisation, parallelism, surrogate usage, and noise handling in a unique way for dealing with simulation-based optimisation problems incurred by these characteristics. In order to handle multiple, conflicting optimisation objectives, the algorithm uses a Pareto approach in which the set of best trade-off solutions is searched for and presented to the user. The algorithm supports a high degree of parallelism by adopting an asynchronous master-slave parallelisation model in combination with an incremental population refinement strategy. A surrogate evaluation function is adopted in the algorithm to quickly identify promising candidate solutions and filter out poor ones. A novel technique based on inheritance is used to compensate for the uncertainties associated with the approximative surrogate evaluations. Furthermore, a novel technique for multi-objective problems that effectively reduces noise by adopting a dynamic procedure in resampling solutions is used to tackle the problem of real-world unpredictability (noise).

The proposed algorithm is evaluated on benchmark problems and two complex real-world problems of manufacturing optimisation. The first real-world problem concerns the optimisation of a production cell at Volvo Aero, while the second one concerns the optimisation of a camshaft machining line at Volvo Cars Engine. The results from the optimisations show that the algorithm finds better solutions for all the problems considered than existing, similar algorithms. The new techniques for dealing with surrogate imprecision and noise used in the algorithm are identified as key reasons for the good performance.

Acknowledgment

I would like to express my gratitude to all those who gave me the possibility to complete this thesis:

- My husband Sanny, for his love and patience. Sanny has not only taken care of the family while I have been spending time on my research studies, but he has also reviewed the thesis and suggested improvements.
- My former colleague Henrik Grimm, for all his help. Henrik's comments and good ideas have been of great value for me.
- My supervisors Philip Moore, Robert John, and Amos Ng, for their support and guidance during my research.
- My colleagues at the Center for Intelligent Automation, Marcus Andersson, Ingemar Karlsson, Martin Andersson, Jacob Svensson, and Nomi Aslam for good company at the office.
- My parents Eva and Lars, and my brother Jonas, for their support. They have helped me not only financially, but also practically with baby sitting, house reparations, horse grooming, house cleaning, dog sitting, and many other things.
- My grandparents Christina, Josef, and Hjördis for their cheerful support.

Table of contents

List of figures

List of abbreviations

APE-GA	Approximate pre-evaluation genetic algorithm
CDR	Confidence based dynamic resampling
EI	Expected improvement
IGD	Inverted generational distance
IHV	Improved hypervolume
IPE-MOEA	Inexact pre-evaluation multi-objective evolutionary algorithm
m-MAES	Modified metamodel-assisted evolution strategy
m-SMS-EMOA	Modified metamodel-assisted S-metric selection evolutionary multi-objective algorithm
MAES	Metamodel-assisted evolution strategy
MOEA	Steady-state ϵ-multi-objective evolutionary algorithm
MOGA	Multiple objective genetic algorithm
MOPSA-EA	Multi-objective parallel surrogate-assisted evolutionary algorithm
NPGA	Niched-pareto genetic algorithm
NSGA-II	Elitist non-dominated sorting genetic algorithm
NSGA-II-ANN	NSGA-II artificial neural network
OSGADO	Objective switching genetic algorithm for design optimisation
PAES	Pareto archived evolution strategy
POI	Probability of improvement
SEAMO	Simple evolutionary algorithm for multi-objective optimisation
SEI	Surrogate error inheritance
SMS-EMOA	Metamodel-assisted S-metric selection evolutionary multi-objective algorithm
SPEA	Strength pareto evolutionary algorithm
SPEA2	Strength pareto evolutionary algorithm 2
SS-NSGA-II	Surrogate-assisted NSGA-II

Chapter 1

Introduction

This chapter provides an introduction to the research area (Section 1.1) and describes the research problem addressed in the thesis (Section 1.2). An overview of the thesis contents is given in Section 1.3.

1.1 Simulation-based optimisation

Real-world problems often contain combinatorial relationships, uncertainty factors, and non-linearities that are too complex to be effectively modelled analytically (April et al., 2004). To illustrate these properties, consider an example of engine crankshaft[1] manufacturing. Specific crankshafts can only be processed in specific machines (combinatorial relationships). Batches of crankshaft variants are prioritised according to a delivery date, but when unpredictable machine breakdowns or unexpected materials' shortages occur (uncertainty factors), machine operators are allowed to change the priorities of the batches on-the-fly. A simple change in order between two batches at one point in the production system may, however, cause indeterminable effects in the system downstream (non-linearities).

[1] A crankshaft is the part of an engine which translates reciprocating linear piston motion into rotation.

For a complex system like this, *simulation-based optimisation* is a powerful alternative to analytical techniques for determining proper system parameters, for example, sizes of buffers storing crankshafts in front of machines (Boesel et al., 2001; Ólafsson and Kim, 2002; April et al., 2004). Simulation-based optimisation is the process of finding the best parameter values for a system, in which the performance is evaluated on the basis of output from a simulated model of the system (Swisher et al., 2000). A *simulation model* essentially consists of a set of system states, and transitions between these states (Fishman, 2001). The evolution of the system is viewed as a sequence of the form

$$s_0, (e_0, t_0), s_1, (e_1, t_1), s_2, ..., \tag{1.1}$$

where s is a system state, e is a system event, and t represents event occurrence time (Fishman, 2001). The above sequence means that the system is started at time 0 in state s_0, taken to state s_1 by event e_0 occurring at time t_0, and so on. The path of how the system evolves in the time process is usually stochastic, in order to capture random fluctuations in real-world systems (Fishman, 2001). Several paths are possible, and which is taken is based on random values of variables in the model.

Simulation-based optimisation is an iterative process; an optimisation algorithm generates a set of input parameter values (referred to as a *solution*) and feeds them to a simulation model, which computes one or multiple performance measures of the system (called *objective values* or *fitness*) (Figure 1.1). In the aforementioned production system, for example, the input parameters might be buffer sizes and the performance measures might be throughput of the system (i.e. production capacity) and work-in-process (i.e. partially completed crankshafts that are somewhere in the manufacturing process and not ready for delivery). Based on the evaluation feedback obtained from the simulation model, the optimisation algorithm generates a new set of parameter values and the generation-evaluation process continues until a user-

defined stopping criterion is satisfied. Such a criterion may, for example, be that a certain amount of time has passed, or that specific objective values have been achieved.

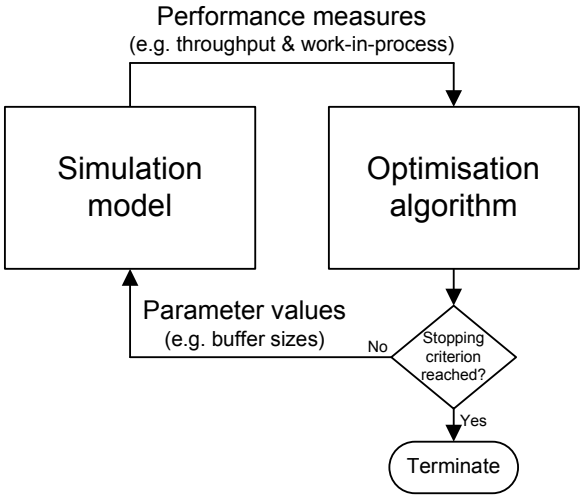


Figure 1.1: Simulation-based optimisation.

The simulation model is a black-box function evaluator in the sense that the relationships between its input parameters and output values are of a closed form (April et al., 2001). As a consequence, there must be a complete separation between the model that represents the system and the algorithm that is used to solve the optimisation problem. An advantage of this separation is that the model of the system can change and evolve, while the optimisation algorithm remains the same (Laguna and Marti, 2003). Another advantage is that the same optimisation algorithm can be used for many systems, since the algorithm does not exploit the internals of the evaluation function (April et al., 2001). A disadvantage, however, is that the optimisation algorithm cannot directly make use of problem-specific information (April et al., 2001).

While traditional, analytical optimisation methods have been unable to cope with the challenges imposed by many simulation-based optimisation problems in an efficient way, such as multimodality, non-separability and high dimensionality (for a description of these concepts, see Appendix A), evolutionary algorithms have been

shown to be applicable to this type of problem (Boesel et al., 2001; Laguna and Marti, 2002; Ong et al., 2004; Emmerich, 2005). *Evolutionary algorithms* are powerful search algorithms developed from biological theories of genetics and reproduction. Since these algorithms do not make explicit assumptions about the underlying structure of the function to be optimised, the black-box nature of the simulation model poses no difficulty (Bäck et al., 1997). In Appendix B, evolutionary algorithms are further discussed and the basic structures of these algorithms are described.

The following section includes a discussion of the challenges involved when applying evolutionary algorithms to real-world problems using a simulation-based optimisation approach. The section also includes a description of the related problems that are addressed in the thesis.

1.2 Problem specification

As discussed in the previous section, evolutionary algorithms are able to deal with complex problems using a simulation-based optimisation approach. Some aspects of these problems must, however, be explicitly handled in the evolutionary algorithm in order to achieve successful results. These aspects include: (*i*) multiple optimisation objectives, (*ii*) high computational cost, and (*iii*) stochastic noise (Evans et al., 1991; April et al., 2001; Laguna and Marti, 2003; Deb, 2004; Jin and Branke, 2005), which are further described below.

Multiple optimisation objectives Almost all real-world problems involve the simultaneous optimisation of multiple objectives, and it is rare for only a single objective to be considered (Zitzler, 1999; Deb, 2004; Mehnen et al., 2004). A *multi-objective problem* is composed of multiple objective functions to be optimised. The difficulty with this type of problem is that there is usually no single optimal solution with respect to all the objectives, as improving the performance of one objective means decreasing the performance of another (Srinivas and Deb, 1995). Instead of a single optimum, there

is a set of optimal trade-offs between the conflicting objectives, called *Pareto-optimal solutions* (Deb, 2004).

In order to manage multiple objectives, specific *multi-objective evolutionary algorithms* have been suggested. Instead of only seeking a single optimum, these algorithms maintain a set of Pareto-optimal solutions. Contrary to many other optimisation techniques, evolutionary algorithms can capture multiple trade-off solutions in a single optimisation run since they maintain a population of solutions.

High computational cost Many real-world problems belong to a class of problems that is called NP-complete, which means that the time required computing an optimal solution increases exponentially with the size of the problem (Cormen and Stein, 2001). NP-complete problems are known to be associated with a high computational cost, since finding an optimal solution requires an exhaustive search. A problem can also be computationally expensive even though it is not NP-complete; real-world optimisation problems in general involve an immense number of possible solutions, and hundreds or thousands of simulation evaluations are needed before an acceptable solution is found (Boesel et al., 2001; Ulmer et al., 2003a). This holds true especially for multi-objective problems, where a significantly larger portion of the search space needs to be explored to obtain a set of Pareto-optimal solutions (Streichert et al., 2005). Even with improvements in computer processing speed, one single simulation may take a couple of minutes or hours of computing time (Boesel et al., 2001; Chafekar et al., 2003). This renders huge optimisation times and poses a serious hindrance to the practical application of evolutionary algorithms in real-world scenarios.

Two techniques that have been suggested for tackling this problem are *parallelism* and *surrogate evaluation functions* (Adamidis, 1998; Emmerich et al., 2006). With *parallel evolutionary algorithms*, multiple processing nodes are used to evaluate several solutions concurrently. Evolutionary algorithms are well suited for parallel environments since solutions in a population can be distributed across processing nodes

and evaluated in parallel without interaction (Adamidis, 1998). A surrogate evaluation function is a computationally cheap approximation of a time-consuming simulation that can be used to estimate the objective values of solutions. By adopting surrogate evaluation functions, the computational time of the optimisation process can be reduced since the computational cost associated with using surrogates is significantly lower than the standard approach of running all evaluations with the simulation model (Jin et al., 2002). However, the imprecision of the surrogate may misdirect the search towards local optima, which must be considered in the optimisation (Ulmer et al., 2003a).

Stochastic noise Most often, real-world problems are subject to stochastic noise as a consequence of uncontrollable variations, caused, for example, by human operators or worn-out machines (Jin and Branke, 2005; Bui et al., 2005; Branke et al., 2007). Noise means that even if the initial conditions of the system and its input parameters are known, the output of the system cannot be predicted and will vary from time to time. These unpredictable variations in the simulation output are harmful to the optimisation process since the evolutionary algorithm can be misdirected to propagate inferior solutions. The common technique for handling this problem is to use the average value obtained from repeated simulations of a solution (Jin and Branke, 2005). Simulating a solution n times reduces the noise by a factor of $\sqrt{n}$, however at the expense of a higher computational cost.

1.2.1 Problem definition

In the previous section, fundamental characteristics of real-world problems were described and different techniques for dealing with them using evolutionary algorithms were outlined. Although these techniques are able to improve optimisation performance, they have only recently gained attention in the context of simulation-based optimisation, and within this area their use is not yet widespread. This holds true

for multi-objective simulation-based optimisation in general (Eskandari et al., 2005), and parallelism, surrogate usage, and noise handling within multi-objective contexts in particular have received limited research attention for evolutionary algorithms in simulation-based optimisation (Yang et al., 2002; Ulmer et al., 2003b,a; Jin and Branke, 2005; Streichert et al., 2005; Bui et al., 2005; Fieldsend and Everson, 2005; Basseur and Zitzler, 2006; Goh and Tan, 2006; Coello Coello et al., 2007; Tan and Goh, 2008). Parallelism, surrogate usage, and noise handling have so far mainly been adopted in single-objective evolutionary algorithms, and comparatively few attempts have been made to incorporate them in multi-objective evolutionary algorithms (Yang et al., 2002; Ulmer et al., 2003b,a; Jin and Branke, 2005; Streichert et al., 2005; Bui et al., 2005; Fieldsend and Everson, 2005; Basseur and Zitzler, 2006; Goh and Tan, 2006; Coello Coello et al., 2007; Tan and Goh, 2008). Further research of the techniques is therefore required to investigate their use in simulation-based optimisation of real-world problems, not only with respect to the individual techniques, but also with respect to their combination. There is currently no evolutionary algorithm that integrates all of these techniques.

1.2.2 Thesis statement

The following hypothesis is defined:

"*An evolutionary algorithm that integrates techniques for multi-objective optimisation, parallelism, surrogate usage, and noise handling can achieve improved optimisation results when undertaking simulation-based optimisation of real-world problems*".

1.2.3 Aim and Objectives

The aim of this thesis is to test the above hypothesis by formulating and evaluating a novel evolutionary algorithm that combines multi-objective optimisation, parallelism, surrogate usage, and noise handling. The following objectives have been identified as

steps in addressing this research study (the chapter that describes the related efforts of each objective is given within parentheses):

O1: Explore the concepts and techniques for dealing with simulation-based optimisation of real-world problems (Chapter 2).

O2: Design and formulate a new evolutionary algorithm for simulation-based optimisation of real-world problems (Chapter 3).

O3: Compare the proposed evolutionary algorithm with existing algorithms (Chapter 4).

O4: Evaluate the proposed evolutionary algorithm on simulation-based optimisation of real-world problems (Chapter 5).

O5: Analyse the proposed evolutionary algorithm on the basis of the evaluation results (Chapter 6).

O6: Propose improvements of, and extensions to, the evolutionary algorithm (Chapter 7).

1.3 Thesis organisation

The thesis is organised as follows. Chapter 2 presents the basic principles of evolutionary algorithms and multi-objective optimisation. Furthermore, the concepts of surrogates are described and an overview of different surrogate techniques is given. The problem of random noise in evolutionary algorithms is also presented, and existing techniques that handle noise in multi-objective optimisation problems are outlined. Based on the concepts presented in the chapter, the design and implementation of a new evolutionary algorithm for addressing real-world problems using a simulation-based optimisation approach is described in Chapter 3. This evolutionary algorithm adopts existing techniques and also adds new concepts in order to efficiently deal

with real-world problems. In Chapter 4, the new evolutionary algorithm is discussed in relation to existing, similar algorithms. Chapter 5 presents an evaluation of the performance of the proposed algorithm when applied to both benchmark problems and complex real-world industrial problems. The results from the evaluation are presented and analysed in Chapter 6. Overall conclusions of the thesis and future work are presented in Chapter 7.

1.4 Summary

This chapter described simulation-based optimisation, a powerful tool to identify the best parameter values for a system. The aim of this thesis is to formulate an evolutionary algorithm that efficiently tackles real-world problems using a simulation-based optimisation approach. In the algorithm, important characteristics of real-world problems are considered, including multiple optimisation objectives, the high computational cost of the optimisation process, and noisy solution evaluations. The next chapter presents techniques for addressing these challenges.

Chapter 2

Background

This chapter describes the fundamentals of multi-objective evolutionary optimisation (Section 2.1), parallelism (Section 2.2), surrogates (Section 2.3), and noise (Section 2.4).

2.1 Multi-objective evolutionary optimisation

Problem solving generally requires the simultaneous optimisation of more than one conflicting objective, especially in real world problems (Zitzler, 1999; Deb, 2004; Mehnen et al., 2004). This section discusses the principles of multi-objective optimisation (Section 2.1.1) and describes five evolutionary algorithms specifically designed for multi-objective problems (Section 2.1.2).

2.1.1 Basic concepts of multi-objective optimisation

As previously discussed in Section 1.2, the difficulty with multi-objective problems is that there is usually no single optimal solution with respect to all objectives, as improving performance for one objective means that the quality of another objective will decrease. Instead, there is a set of optimal trade-offs between the conflicting objectives, known as the Pareto-optimal solutions or the Pareto front (Deb, 2004). Figure 2.1 illustrates the Pareto concepts for a minimisation problem with two objectives f_1

and f_2. In this example, solutions *A*-*D* are *non-dominated*, that is, Pareto optimal, since for each of these solutions there is no other solution that is superior in one objective without being worse in another. Solution *E* is *dominated* by *B* and *C* (but not by *A* or *D*, since *E* is better than these two in f_1 and f_2, respectively). Different *Pareto ranks* can also be identified among solutions. Rank 1 includes the Pareto-optimal solutions in the complete population, and rank 2 the Pareto-optimal solutions identified when temporarily discarding all solutions of rank 1, and so on.

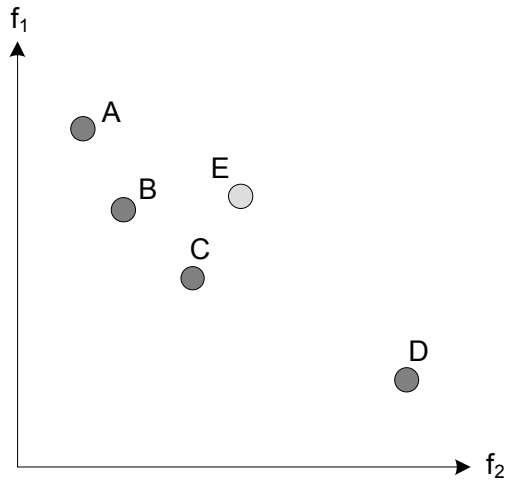


Figure 2.1: Non-dominated and dominated solutions.

As an example of a multi-objective problem, consider the configuration of buffers between machines in the crankshaft production system introduced in Chapter 1. It is desirable to find a buffer configuration that maximises the overall throughput of the system, since with a high throughput rate, improvements in productivity and profitability can be achieved. At the same time, it is also desirable to minimise the amount of work-in-process in the system in order to avoid capital being tied up in partially completed crankshafts. With large buffers, the throughput increases since machine starvation is avoided and the impact of the interference caused by variability in processing times decreases. However, with large buffers the amount of work-in-process also increases since a large number of crankshafts are kept in the system (assuming that the system is working at full capacity and buffers are constantly loaded

with new raw materials). Conversely, small buffers decrease both the work-in-process and the throughput. This trade-off problem between throughput and work-in-process is illustrated with an example in Figure 2.2. In this example, there are five solutions to the problem, none of which is superior to any other when considering both objectives at the same time (assuming that both objectives are equally important for the decision maker).

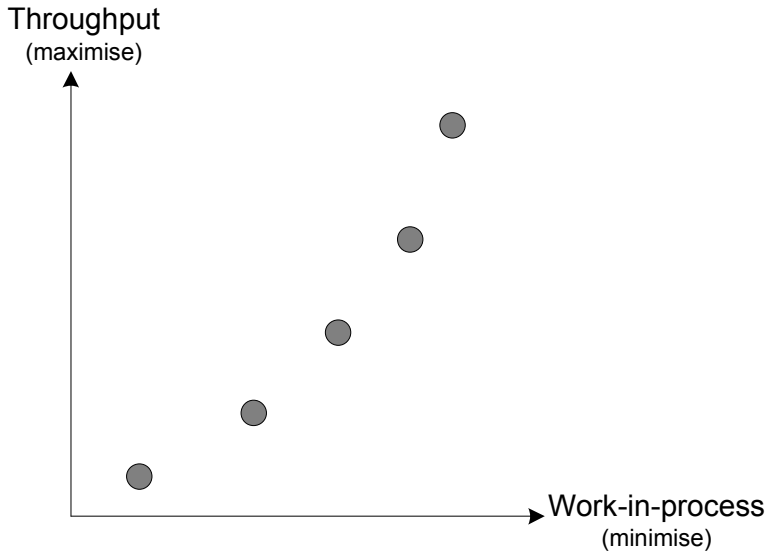


Figure 2.2: Example of trade-off solutions in multi-objective optimisation.

Although it is important to have as many (trade-off) optimal solutions as possible in multi-objective optimisation, the user needs only one solution regardless of the number of objectives (Deb, 2004). Which of the optimal solutions should be chosen is up to the user based on previous experiences and qualitative information (for example, ergonomic conditions or set-up of factory workers for the day). The process of choosing a solution is illustrated in Figure 2.3 (adopted from Deb 2004). With a single-objective problem, only one solution will be found in step 2 and the subsequent steps are of no relevance.

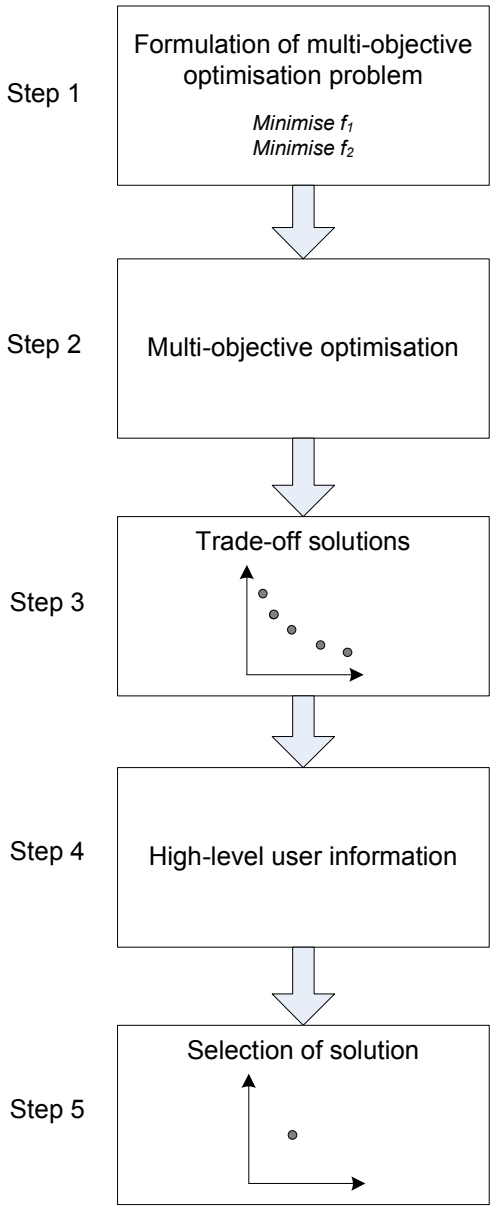


Figure 2.3: Multi-objective optimisation procedure.

Evolutionary algorithms are very well suited for handling multi-objective problems (Deb, 2004). Since evolutionary algorithms maintain a population of solutions, it is possible to find multiple Pareto-optimal solutions in a single optimisation run. In comparison, classical optimisation algorithms are considerably less efficient since they can only find one Pareto-optimal solution in each run. In the following section, an overview of the most salient multi-objective evolutionary algorithms is presented.

2.1.2 Overview of multi-objective evolutionary algorithms

The first evolutionary algorithm specifically designed for multi-objective optimisation, called vector-evaluated genetic algorithm, was proposed by Schaffer (1984). Since then, a number of studies of multi-objective evolutionary algorithm have been presented. This section presents an overview of five of the most important multi-objective evolutionary algorithms: multiple objective genetic algorithm, niched-pareto genetic algorithm, elitist non-dominated sorting genetic algorithm, strength pareto evolutionary algorithm, and pareto archived evolution strategy (Srinivasan and Rachmawati, 2006). A detailed review of extensions of these algorithms is presented in Chapter 4.

Multiple objective genetic algorithm

The *multiple objective genetic algorithm* (MOGA) is based on a genetic algorithm (Fonseca and Fleming, 1993). For a description of genetic algorithms, see Appendix B. When assigning fitness, MOGA makes use of non-dominated sorting in combination with a measure for estimating the diversity among solutions. Each solution p is assigned a rank r corresponding to the number of solutions that dominate it (n), plus one:

$$r_p = n + 1. \tag{2.1}$$

The maximum number of ranks equals the population size N. However all ranks between $1 - N$ are not necessarily assigned since two or more solutions may have the same rank. To maintain diversity among solutions with the same rank, so called *niching* is used. The idea of niching is to punish solutions that are close to each other in the same local optimum in the search space. To estimate the number of solutions belonging to the same optimum a sharing function is used (Equation 2.2).

$$sh(d) = \left\{ \begin{array}{cc} 1 - \left(\frac{d}{\sigma}\right)^{\alpha} & if\ d \leq \sigma \\ 0 & otherwise \end{array} \right\} \tag{2.2}$$

In Equation 2.2, d is the Euclidean distance between two solutions, and σ is the user-defined maximum distance between two solutions in the same optimum. The sharing function takes a value in the interval [0,1]. If two solutions are identical, their Euclidean distance will be zero and the function value will be 1. If, instead, their Euclidean distance equals, or is greater than, σ (i.e. $d \geq \sigma$) the function value will be 0. The parameter α is used to weight the sharing effect. If α takes a value of 1, the effect is linearly reduced from one to zero. The sharing function is used to determine the crowding around a solution, called the niche count. The niche count for a solution p is calculated according to Equation 2.3

$$nc = \sum_{j=1}^{R} sh\left(d_{pj}\right), \tag{2.3}$$

where R is the number of solutions with the same rank as p. A fitness value for s is calculated by dividing the value of r_p by the value of nc_p. The idea of this fitness assignment formula is to make sure that, within the same rank, a larger selection pressure will be put on solutions in less crowded areas. In this way, a high diversity in the population can be maintained. The fitness assignment procedure is rather simple, which is one of the main advantages of the algorithm (Deb, 2004). However, a drawback of the procedure is that it does not guarantee that a solution in a worse rank always has worse fitness than every solution in a better rank, which might lead to slow convergence (Deb, 2004).

Niched-pareto genetic algorithm

The *niched-pareto genetic algorithm* (NPGA) uses an updated niching strategy for the selection of solutions reproducing (Horn et al., 1994). Unlike most of the other multi-objective evolutionary algorithms, which use a proportional selection operator, this algorithm uses binary tournament selection. Two solutions, i and j, are randomly chosen from the parent population to compete in the tournament. A sub-population of size t is randomly chosen from the parent population and i and j are checked for

domination against each solution in the sub-population. If either i or j dominates all solutions in the sub-population but the other does not, the dominant one is chosen as the winner of the tournament. If both i and j are either dominated or not dominated by any solution in the sub-population, they are checked against the (partially filled) offspring population. This is done by placing i and j in the offspring population and calculating a niche count. The solution with the smallest niche count is the winner of the tournament. With this tournament selection strategy, there is no need to specify the fitness values of solutions. This is an advantage since it eliminates the subjectivity involved in fitness assignment. However, a drawback is that a new user-defined parameter t is introduced, which highly influences the performance of the algorithm (Deb, 2004).

Elitist non-dominated sorting genetic algorithm

In the *elitist non-dominated sorting genetic algorithm* (NSGA-II), the selection of solutions for the next generation is done from the set R, which is the union of the parent population and the offspring population (both of size N) (Deb et al., 2000). Non-dominated sorting is applied to R and the next generation of the population is formed by selecting solutions from one of the Pareto fronts at a time. The selection starts with solutions in the best Pareto front, then continues with solutions in the second best front, and so on, until N solutions have been selected. If there are more solutions in the last front than there are remaining to be selected, niching is applied to determine which solutions should be chosen. All the remaining solutions are discarded. The selection procedure is illustrated in Figure 2.4 (adopted from Deb 2004).

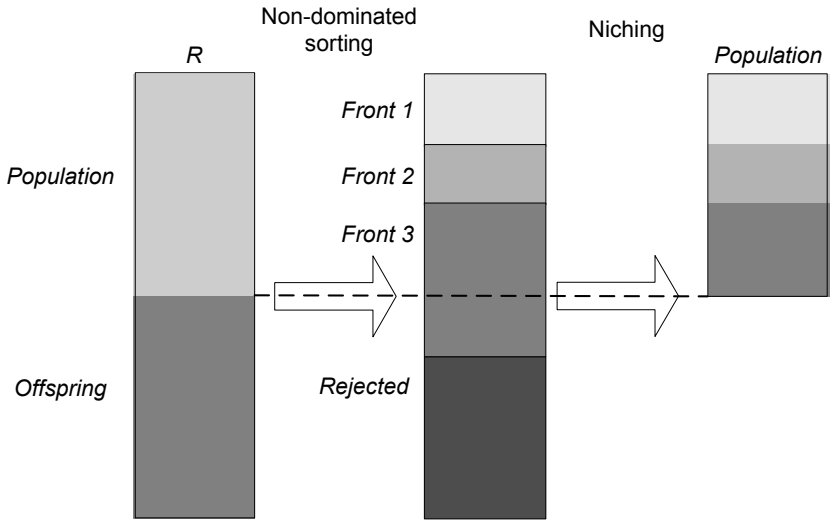


Figure 2.4: Selection procedure of NSGA-II.

In the parental tournament selection, the highest ranking solution located in the least crowded area is the one chosen for mating. The crowding comparison element eliminates the extra niching parameter used in MOGA and several other multi-objective algorithms, which is an advantage of NSGA-II. However, a disadvantage is that when there are more than N solutions in the first non-dominated front, some crowded Pareto-optimal solutions may be discarded in favour of other non-dominated, yet not Pareto-optimal solutions (Deb, 2004).

Strength pareto evolutionary algorithm

The main characteristic of *strength pareto evolutionary algorithm* (SPEA) is that it stores all Pareto-optimal solutions in an external set (Zitzler and Thiele, 1998). The idea of this approach is that the population size should not restrict the number of Pareto-optimal solutions and no Pareto-optimal solutions should be lost during the optimisation. However, the performance of the algorithm is highly dependent on the balance between the population size and the size of the external set, and this is one of the main disadvantages of the algorithm (Deb, 2004).

In an iteration of the algorithm, the external Pareto set is first updated. This is done by copying all non-dominated solutions in the population to the external Pareto set,

followed by a deletion of solutions in the sets that become dominated (if any). If the number of individuals in the external Pareto set exceeds the user-defined maximum, the set is reduced using a clustering technique based on Euclidean distances. The clustering algorithm is parameterless and ensures a good spread among solutions, which is an advantage of the algorithm. From the union of the population and the external Pareto set, solutions are randomly chosen for reproduction. In Figure 2.5 (adopted from Zitzler and Thiele 1998), the overall procedure of the algorithm is illustrated.

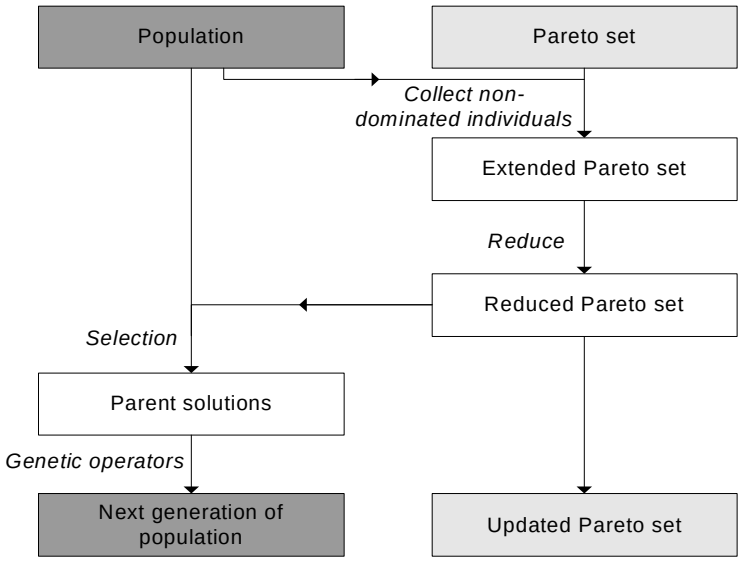


Figure 2.5: Overall procedure of SPEA.

An improved version of algorithm has been suggested, called SPEA2 (Zitzler et al., 2001). The main differences between the original algorithm and the improved version are:

- an improved archive reduction method that always preserves boundary solutions,
- incorporation of information about how many other solutions are being dominated by a certain solution, and how many solutions are dominating it, and
- incorporation of crowding information in the fitness assignment.

Pareto-archived evolution strategy

The *pareto archived evolution strategy* (PAES) is based on an evolution strategy, in contrast to the previously described multi-objective algorithms which are based on a genetic algorithm (Knowles and Corne, 2000). For a description of evolution strategies, see Appendix B. In its basic form, PAES uses a (1+1)-evolution strategy, which is the simplest form of an evolution strategy. This strategy applies a local search and selection takes place between one parent and one offspring. Mutation is applied to a single parent to create a single offspring and the parent of the next generation is the one having the best fitness. Similar to SPEA, an external archive of the best solutions found so far is maintained. If an offspring dominates its parent, it is included in the archive. If the parent and the offspring are mutually non-dominating, the offspring is compared to members of the archive and if it is dominated by any solution in the archive, it is discarded and a new offspring is created. If it instead dominates, or is not dominated by, any solution in the archive, it might become the parent of the next generation and be included in the archive, depending on the availability of slots. Since the number of slots in the archive is limited, a density calculation takes place if there are no free slots. In this density calculation, the objective space is divided into a number of multi-dimensional boxes (called hypercubes), as illustrated in Figure 2.6 (adopted from Deb 2004) for a two-dimensional problem. The number of archived solutions in each hypercube is counted and if the offspring is located in a less crowded hypercube than its parent (which is already in the archive), it becomes the parent of the next generation. It is also included in the archive as long as it is not located in the most crowded hypercube.

The direct control of diversity that is achieved with this approach is one of the main advantages of the algorithm (Deb, 2004). However, it might be difficult to find an appropriate size of the hypercubes, especially since the size of the area in the search space in which solutions are scattered changes during the search (Deb, 2004).

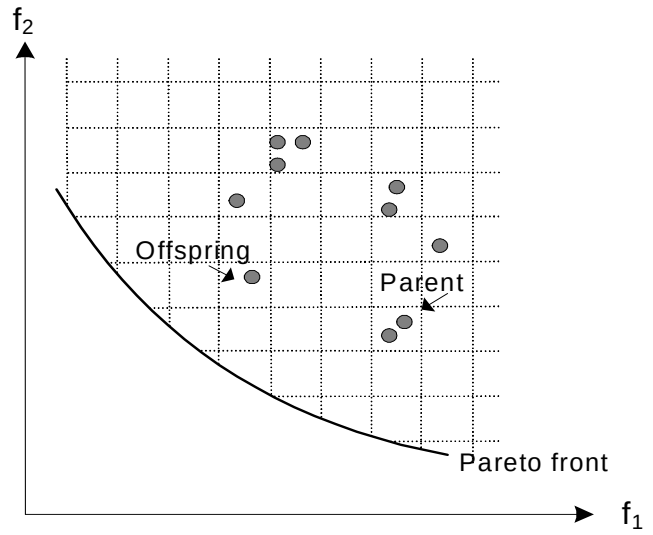


Figure 2.6: Offspring located in a less crowded hypercube than its parent.

(1+λ) and (μ+λ) variants of the PAES have been suggested as extensions to the basic algorithm (Knowles and Corne, 2000). In the (1+λ)-variant, λ offspring are created from the parent and compete to become the parent of the next generation. The (μ+λ)-variant is a global search algorithm in which λ offspring are generated from μ parents. The selection of μ solutions from the combined set of the λ offspring and the μ parents is based on a dominance score. This score is calculated as follows: If a solution dominates one or several members of the archive, it gets a score of 1. If it is non-dominated, it is assigned a score of 0. If it is dominated by any member in the archive, it gets a score of -1. Based on the dominance scores, in combination with the density measure described earlier, fitness is assigned to the solutions and used in the selection.

2.2 Parallelism

In the context of optimisation, parallelism refers to the concurrent evaluation of solutions of multiple processing nodes. Evolutionary algorithms are generally easy to parallelise since solutions in a population can be distributed across processing nodes and evaluated in parallel without interaction. In addition to time reduction,

parallel evolutionary algorithms also have other advantages such as improvement of population diversity and ease of working together with another algorithm in parallel (Jaimes and Coello Coello, 2005). In multi-objective evolutionary optimisation, there are four major parallelisation models: master-slave, island, diffusion, and hybrid (Coello Coello et al., 2007). These are described below.

2.2.1 The master-slave model

With the *master-slave model*, a central master node stores the population and performs evolutionary operations, while solution evaluations are distributed over multiple slave nodes (Figure 2.7) (Veldhuizen et al., 2002). The model is called synchronous if the master waits for the slaves to finish evaluating an entire generation before generating any solutions of the following generation, and asynchronous if the master does not wait. Since there is a lot of communication between the master and the slaves, the solution evaluations must be significantly more time-consuming than the communication for the model to be efficient. In real-world applications, this is generally not a problem since the communication overhead is usually negligible in comparison to the cost of evaluating a solution.

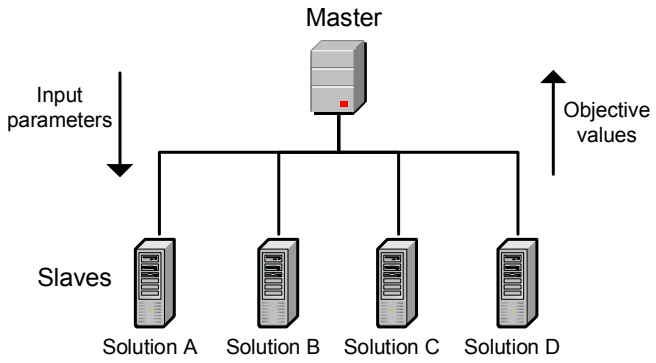


Figure 2.7: Master-slave model.

An example of a multi-objective evolutionary algorithm using the master-slave model is the *parallel multi-objective particle swarm algorithm* (Mostaghim et al., 2008). This algorithm is something in between synchronous and asynchronous; the master

distributes solution evaluations to its n slaves and waits until m results have been sent back ($m < n$). When m results have been received, the master generates a new generation of the population and sends this to available slaves for evaluation. The idea of letting the master continue the evolution although all slaves have not finished their evaluations is to prevent slow slaves delaying the optimisation process.

2.2.2 The island model

With the *island model*, independent sub-populations are run on multiple processing nodes (Figure 2.8) (Veldhuizen et al., 2003). Every island can either have the same parameter settings and evolutionary operators (homogeneous approach), or these can vary between islands (heterogeneous approach). After every n generation the best m solutions are migrated from one sub-population to another. Migrations can be done through the master or directly between sub-populations. Compared to the master-slave model, this parallelisation model requires considerably less communication.

For multi-objective evolutionary algorithms, a "divide and conquer" variant of the island model has been suggested. In this variant, individual sub-populations specialise in one or several given objectives during the optimisation. The idea is that these sub-populations should focus on certain areas of the Pareto front and thus become more efficient. However, finding a suitable partitioning of a given optimisation problem requires a priori knowledge of the topology of the search space and the shape of the Pareto front.

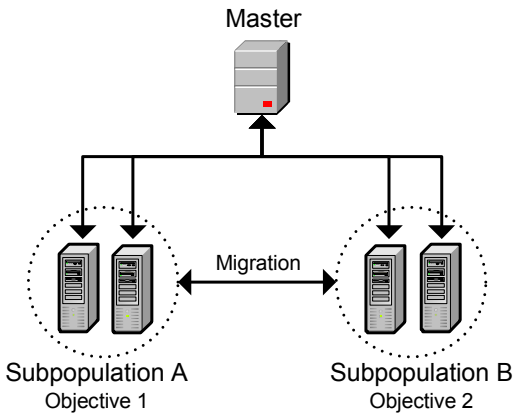


Figure 2.8: Island model.

An example of a island-based multi-objective evolutionary algorithm is the *parallel strength pareto multi-objective evolutionary algorithm* (Xiong and Li, 2003). In this algorithm, islands can be either generational or steady-state. Each island evolves a sub-population of solutions with different crossover and mutation probabilities, but all use the same tournament selection operator. Population members are exchanged between islands with a user-defined dynamic migration frequency.

An example of an algorithm using the "divide and conquer" variant of the island model is the *parallel single front genetic algorithm* (de Toro Negro et al., 2004). In this algorithm, islands are assigned separate objectives to optimise, and at regular intervals solutions are sent from the islands to the master. The master sorts the solutions and checks if they should be included in an external archive of Pareto-optimal solutions.

2.2.3 The diffusion model

The *diffusion model* is similar to the master-slave model in that it deals with one population, but mating partners are selected among a few neighbours directly reachable by the network topology (in contrast to the other models in which the selection pool is composed of the whole population or sub-population) (Streichert et al., 2005). The neighbourhood region of a solution could have the shape of a square, rectangle, cross, or any other form (Figure 2.9). Through overlapping or dynamically changing

neighbourhoods, a slow diffusion of information through the population takes place. The diffusion model involves a substantial communication cost within the neighbourhood and is designed mainly for massively parallel environments with fast, local communication networks.

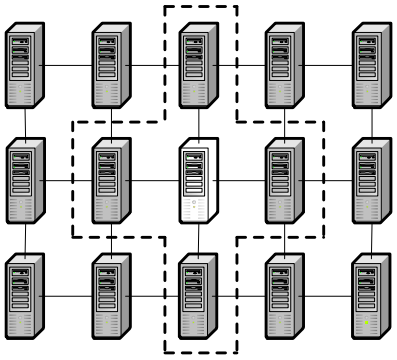


Figure 2.9: Diffusion model, cross-formed neighbourhood region with distance = 1.

An example of a diffusion model is the *cellular genetic algorithm* (Nebro et al., 2007). In this algorithm, square lattice geometry is used with small overlapping neighborhoods. For each population member p_i, a random neighbour is selected for mating. The offspring p_j created with p_i as parent replaces p_i if $p_j > p_i$, otherwise it is discarded.

2.2.4 The hybrid model

The *hybrid model* has been proposed as an extension of the island model (Cantú-Paz, 2000). The sub-population at each island is evolved by means of one of the previously described parallelisation models, that is,

(a) each island contains a master-slave-based evolutionary algorithm, or

(b) each island contains an island-based evolutionary algorithm, or

(c) each island contains a diffusion-based evolutionary algorithm (Figure 2.10).

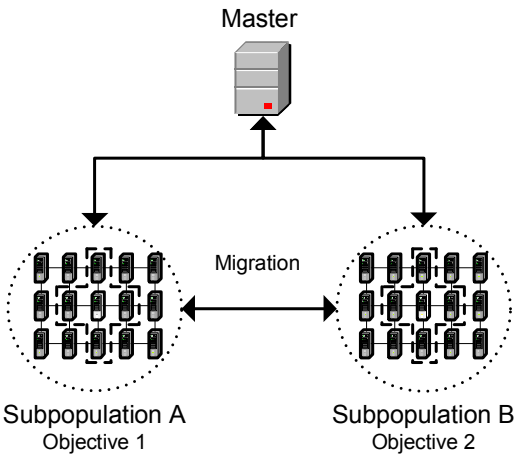


Figure 2.10: Hybrid of island model and diffusion model.

A detailed literature search could not find an implementation of the hybrid model in the context of multi-objective evolutionary algorithms.

2.3 Surrogate evaluation functions

To increase the effectiveness of multi-objective evolutionary algorithms, especially when applied to real-world problems involving time-consuming simulations, the incorporation of surrogate evaluation functions has been suggested. In this chapter, the concepts of surrogate evaluation functions are described (Section 2.3.1) and different techniques for their construction are presented (Section 2.3.2 and Section 2.3.3).

2.3.1 Basic concepts

A surrogate evaluation function (or simply a surrogate) is a computationally cheap simplification of the simulation. Although the surrogate is less accurate than the simulation, it often has satisfactory estimation capabilities to assist in the optimisation process (Jin et al., 2003; Hüsken et al., 2005). However, since the surrogate is imprecise, it cannot be used alone in the optimisation process, but must be used in conjunction with the simulation. Different approaches for how to combine a simulation and a

surrogate have been suggested, and from an overall perspective these approaches are either generation-based or individual-based (Jin, 2005). The idea of the *generation-based* approach is that in every cycle of m generations, the simulation is used to evaluate n generations (called *controlled generations*). In Figure 2.11 (adopted from Jin et al. 2002), an example of a generation cycle is given. In the *individual-based* approach, a certain number of individuals in each generation is evaluated using the simulation (called *controlled individuals*). It can be noticed that, in contrast to the generation-based approach, the controlled individuals do not necessarily belong to the parent population (they may, for example, belong to the set of offspring).

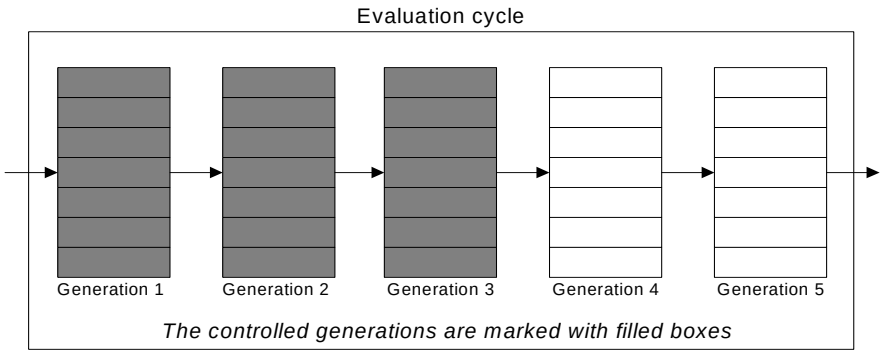


Figure 2.11: Generation-based approach.

Although the use of surrogates might decrease the computational time of the optimisation process, it is important to keep in mind that surrogates are imprecise by nature (Jin, 2005). This imprecision may mislead the optimisation algorithm into propagating inferior solutions; weak offspring might be chosen for the next generation while good ones are excluded. When this happens, the search may converge to a false optimum (Ulmer et al., 2003a; Ong et al., 2004; Jin, 2005; Büche et al., 2005; Lim et al., 2005). For successful results, the imprecision of the surrogate must therefore be considered in the optimisation, otherwise a poor convergence is likely (Ulmer et al., 2003a; Büche et al., 2005; Lim et al., 2005).

For the construction of surrogates, various techniques exist. In general terms, these techniques can be divided into two categories: *surrogate approximations* and *surrogate*

models. While the former treat the simulation as a black box, knowing nothing about its inner workings, the latter treat the simulation as a white box and explicitly attempt to imitate its internals. Surrogate approximations and surrogate models are further described in Section 2.3.2 and Section 2.3.3, respectively.

2.3.2 Surrogate approximations

Surrogate approximations (also called *metamodels*) make estimations based on data samples obtained from previous runs of the simulation. More precisely, a surrogate approximation is trained to learn the functional relationship between the output y and the input x of the simulation. If the simulation is represented as $y = f(x)$, then a surrogate approximation is represented as $\hat{y} = f(x) + e(x)$, where e is the approximation error. From the outside, the surrogate approximation is a black-box, that is, only its input x and output $\hat{y}$ are observed and analysed. The estimation accuracy of a surrogate is dynamic and will change when new data samples are added or removed. Although training of a surrogate approximation takes some time, it is reasonable to assume that this cost is negligible compared to the cost of a simulation evaluation, especially in real-world problems (Ratle, 1999; El-Beltagy et al., 1999; Giannakoglou, 2002; Emmerich, 2005).

The use of surrogate approximations to reduce the limitations of time consuming simulations was first proposed in Blanning (1975). Since then, a variety of different surrogate approximation techniques have been proposed. Comparative studies have shown that there is no universally superior approximation technique, but the performance of the different techniques depends on the characteristics of the problem under consideration (Jin et al., 2001; Queipo et al., 2005). In the remainder of this section, three of the most popular techniques in multi-objective evolutionary optimisation are described to exemplify surrogate approximations: artificial neural networks, Kriging, and radial basis function networks.

Artificial neural network

In general terms, an *artificial neural network* is a non-linear statistical data modelling method used to model complex relationships between inputs and outputs (Mehrotra et al., 1996). The inspiration for the technique originated from the area of neuroscience and the study of the neurons as information processing elements in the central nervous system. Essentially, artificial neural networks are simple mathematical models defining a function $f : X \rightarrow \widehat{Y}$. The function $f(x)$ is defined as a composition of a number of other functions $g_i(x)$, which can also be defined as a composition of a number of other functions, altogether represented as a network structure. A variety of different types of artificial neural network architectures exists. The one most commonly used is the feedforward artificial neural network, in which the information only moves forward from the input nodes, through the hidden nodes and to the output nodes via a series of weights (Villaseñor et al., 2006). The sum of the products of the weights and the inputs is calculated in each node, and if the value in an output node is above some threshold the node is activated. The activation is determined by an *activation function*, such as the sigmoid function or the binary threshold function. An example of a feedforward artificial neural network is shown in Figure 2.12.

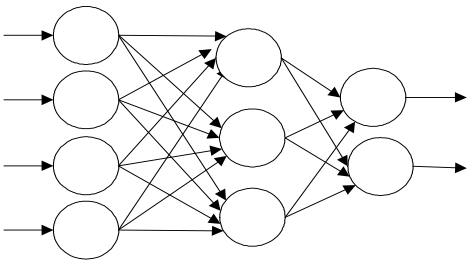


Figure 2.12: Feedforward artificial neural network.

There are a variety of learning algorithms for training artificial neural networks. The most popular of these is *backpropagation* (an abbreviation for "backwards propagation of errors"), a supervised learning technique (Mehrotra et al., 1996). In supervised learning, a set of input-output samples (x, y) is given and the aim is to find a function f that matches the examples. In backpropagation, the output values of the network

are compared with the correct answer to compute the value of an error function. A commonly used error function is the *mean squared error* which tries to minimise the average error between the output of the network, $\widehat{y}$, and the target value y for all example samples N, according to Equation (2.4).

$$\frac{1}{N}\sum_{i=1}^{N}\left(\widehat{y}_i - y_i\right)^2 \tag{2.4}$$

As the name of the backpropagation algorithm implies, the errors are propagated backwards from the output nodes to the inner nodes. Using the error information, the weights of all connections are adjusted using a gradient descent method to reduce the value of the error function. More precisely, the derivative of the error function with respect to the network weights is calculated and the weights are changed such that the error decreases. For this reason, the backpropagation algorithm requires that the transfer function used by the neurons is differentiable. A commonly used transfer function is the sigmoid function, defined by

$$\frac{1}{1+exp^{-x}}, \tag{2.5}$$

where x is the summed input of the network. When the process of weight adjustment has been repeated for a sufficiently large number of training cycles, the artificial neural network will (hopefully) converge to a state where its error is small. It is assumed that the training process will enable the network to generalise to new situations and predict the correct output for samples not presented during training. When there are insufficient training samples, or the performance of the learning process is too long, the generalisation capability of the artificial neural network may become poor. With too much training, the artificial neural network memorises input-output relationships of the training samples and fails to capture the true nature of the target function. This problem is called *overfitting* and means that the performance on training samples increases while the performance on unseen samples becomes worse.

To avoid overfitting, *cross-validation* can be used to indicate when further training will not result in improved generalisation (Mehrotra et al., 1996). In cross-validation, some of the data samples from the training set are set aside and used to test the performance of the network on new data. An example of the concept of cross-validation is shown in Figure 2.13 (training error of the artificial neural network is shown in black and cross-validation error in gray). The network has become overfitted at the point when the validation error increases while the training error steadily decreases (the gray line).

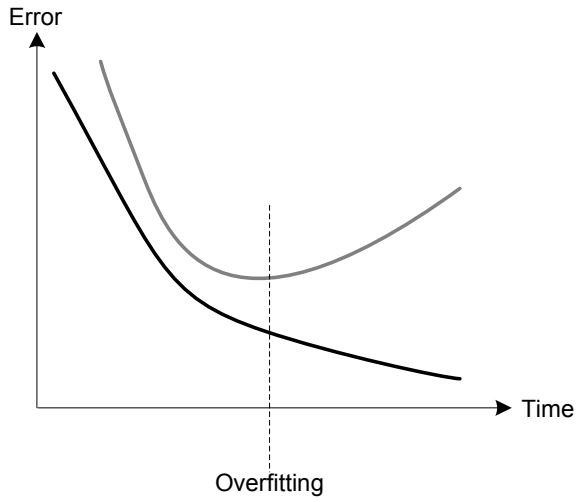


Figure 2.13: Cross-validation.

Kriging

Kriging is an interpolation method named after the mining engineer Daniel Gerhardus Krige who developed the technique in the 1960s to predict ore concentration in mines (Krige, 1966). In the literature, Kriging models are also known as *Gaussian Random Field models*, *Gaussian processes*, and *Gaussian random functions* (Emmerich, 2005). Kriging belongs to a family of interpolation methods called *geostatical methods* (Simpson et al., 1998). The basic idea of these methods is that weights surrounding measured points can be used to derive predictions for unmeasured points. The general formula used for interpolation is

$$\widehat{Z}(s_0) = \sum_{i=1}^{N} \lambda_i Z(s_i), \tag{2.6}$$

where $Z(s_i)$ is the measured value of point i, λ_i the weight of point i, s_0 is the point whose value is to be predicted, and N is the number of measured points. In traditional interpolation methods (such as the Spline method), the value of λ_i depends only on the distance to the value to be predicted. In Kriging, however, the weights are not only based on the distance to the value to be predicted, but also on the spatial relationships among the measured points (Simpson et al., 1998). Points that are located near the value to be predicted are considered to have a higher degree of spatial correlation than points far away. Using correlation information, a Kriging model can not only provide a prediction, but also a confidence measure of this prediction. An example of Kriging interpolation and confidence interval is given in Figure 2.14.

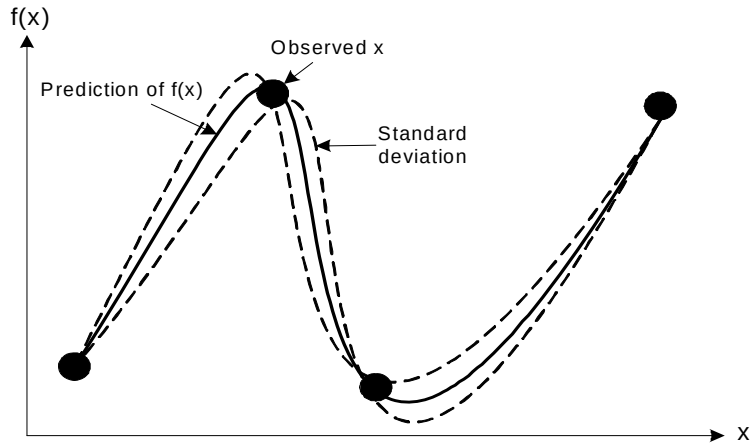


Figure 2.14: Kriging model.

A Kriging model can be seen as a combination of a global model plus a localised deviation:

$$\widehat{y}(x) = g(x) + Z(x), \tag{2.7}$$

where g represents the global model, and Z is a Gaussian random function with mean zero and non-zero covariance represented by

$$cov\left[Z\left(x_i\right), Z\left(x_j\right)\right] = \sigma^2 R\left(x_i, x_j\right), \tag{2.8}$$

where σ^2 is the variance of Z, and R is the spatial correlation between two points i and j (Jin, 2005). Several different correlation functions exist, from which the Gaussian correlation function (Sacks et al., 1989) is the most popular (Jin, 2005).

The main advantage of Kriging is that it has less parameters compared to artificial neural networks. For example, no network size or architecture needs to be specified. The major disadvantage is that model construction and prediction is very time-consuming, especially when the number of data points and variables grow (Jin et al., 2001; Jin, 2005; Büche et al., 2005; Voutchkov and Keane, 2006). With N data points, the computational complexity of constructing the model is of order N^3, the prediction of the value of a new point is of order N, and the prediction of the standard deviation is of order N (El-Beltagy et al., 2001; Büche et al., 2005).

Radial basis function network

A *radial basis function* is a function whose value depends on the distance from a centre point c (Baxter, 1992). A *radial basis function network* is an approximation of the form

$$\widehat{y}(x) = \sum_{i=1}^{N} w_i \rho_i(||x - c_i||) \tag{2.9}$$

represented by the sum of N radial basis functions. An radial basis function ρ_i is associated with a centre c_i, and a weight w_i (Baxter, 1992). Usually, the given training samples are used as the centres of the radial basis functions (Büche et al., 2005).

A radial basis function network can be seen as an artificial neural network which uses radial basis functions as activation functions (Mehrotra et al., 1996). Typically, a radial basis function network has three layers: an input layer, a hidden layer with non-linear radial basis function activation functions, and a linear output layer (Figure 2.15) (Mehrotra et al., 1996). Returning to Equation 2.9, N is then the number of neurons

in the hidden layer, w_i is the weight from neuron i to the output neuron, and c_i is the centre for neuron i. The radial basis functions ρ, used for the activation function in the hidden nodes, are usually Gaussian basis functions (Mehrotra et al., 1996).

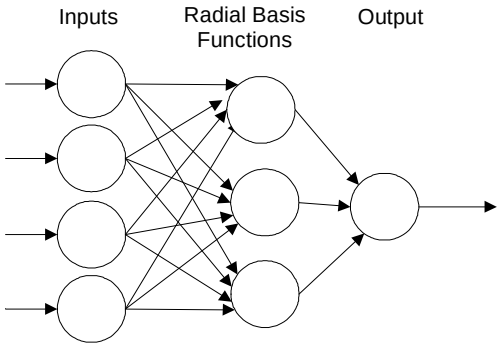


Figure 2.15: Radial basis function network.

There are three types of parameters of a radial basis function network that need to be adapted through training: the centres c_i, the output weights w_i, and the basis widths β_i used in the hidden nodes. The most common function for training radial basis function networks is the *least squares function* (Mehrotra et al., 1996), in which the summed square of the difference between the surrogate output $\widehat{y}$ and the simulation output y is:

$$\sum_{i=1}^{N} \left(\widehat{y}_i - y_i\right)^2. \tag{2.10}$$

The least square value can be minimised by, for example, using gradient descent training (Mehrotra et al., 1996). With gradient descent training, the weights are adjusted by changing them in a direction opposite from the gradient of the objective function.

In general, radial basis function networks have fast learning speed and global generalisation power (Tsao, 2007). However, similar to artificial neural networks, there is a large amount of trail-and-error associated with the construction and training of radial basis function networks (Tsao, 2007).

2.3.3 Surrogate models

Surrogate approximations have been used with great success in many optimisation problems. However, in some situations constructing and using surrogate approximations can be next to impossible. This might be the case when the number of simulation inputs is too large and/or varies between solutions. For example, Andersson et al. (2007) describe an optimisation problem of batch scheduling in a complex production line in the automotive industry. The schedule to be optimised in this problem describes, in detail, how the production should be organised; including division of products into batches, batch sizes, ordering of batches, and machine operations for different batches (Andersson et al., 2007). The complex nature of the schedules in combination with a vast number of products means that hundreds of simulation inputs are necessary to represent a schedule. The exact number of inputs needed varies between schedules, since the configuration of batches and their processing is flexible. Constructing a surrogate approximation for a problem like this is very difficult. Surrogates representing problems of high dimensionality (i.e. large number of inputs) are generally hard to construct, especially in real-world scenarios with a limited number of training samples, due to the high computational cost of generating the samples (Ulmer et al., 2003a; Ong et al., 2004; Jin, 2005; Büche et al., 2005; Lim et al., 2005). The problem becomes even more difficult when the number of inputs varies between schedules.

When constructing a surrogate approximation is difficult, a surrogate model may be an alternative. A surrogate model solves the same problem as the simulation but makes a number of simplifications (Giunta and Watson, 1998). The model tries to imitate the simulation using a white-box approach (in contrast to surrogate approximations, which use a black-box approach). A surrogate model is less complex than a simulation, and also computationally cheaper. In contrast to surrogate approximations, a surrogate model is static in the sense that it does not change during the optimisation (i.e. it is not trained).

An example of the use of a surrogate model is given in Persson et al. (2006). They considered the problem of finding optimal mail operation schedules in an automatic mail sorting facility. An operation schedule defines the mail batches to be sorted, the different machines to be used for each batch, the operations to perform, and the start time of each machine. The large number of batches to be scheduled in combination with the extensive amount of information for each batch means that a surrogate model is better suited to this problem than a surrogate approximation. The surrogate model is constructed using the C# programming language based on the same principles as the simulation model, but with a number of simplifications. For example, stochastic events in the system are not considered in the surrogate model. In the optimisation, the surrogate model is used to estimate sorting deadlines and check if the schedule is valid. If a solution is considered invalid by the surrogate model, it is not sent to the simulation for further evaluation, and, in this way, the computational time of the optimisation process can be reduced.

2.4 Noise

This section describes how evolutionary algorithms are influenced by noise (Section 2.4.1) and presents existing techniques for handling noise in multi-objective optimisation problems (Section 2.4.2).

2.4.1 The effects of noise in evolutionary algorithms

A certain degree of randomness, so called *noise*, is an inherent property of most real-world systems. Returning to the production system outlined in Chapter 1, noise in a system like this may be caused by unpredictable machine breakdowns due to worn-out machine parts, or by the fluctuating skills of machine operators on different shifts. When a system is subject to noise, repeated evaluations of the same solution over time will result in different objective values. This effect is exemplified in Figure

2.16 (adopted from Büche et al. 2002), where the objective value returned from the evaluation function f has an error that is governed by a normal distribution and therefore varies from time to time.

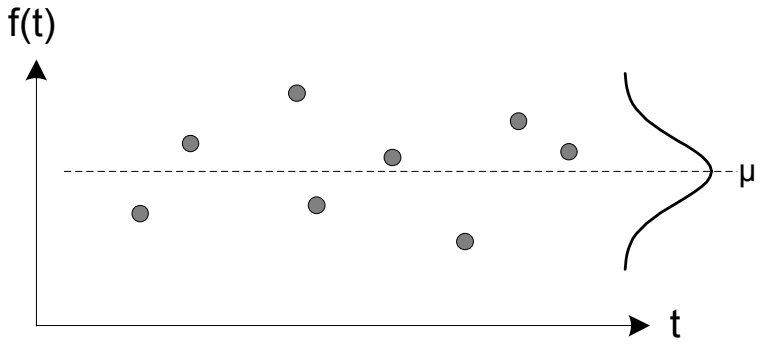


Figure 2.16: Varying fitness values due to noise.

A noisy evaluation function is also illustrated in Figure 2.17 (adopted from Pietro et al. 2004). In this figure, the function to be optimised is shown without noise to the left and with added noise to the right (the probability of a function evaluation resulting in a particular value is represented by the shading; the darker the area the more likely the value occurs). While it is trivial to optimise the original function by an evolutionary algorithm, the problem becomes significantly harder to solve when noise is present.

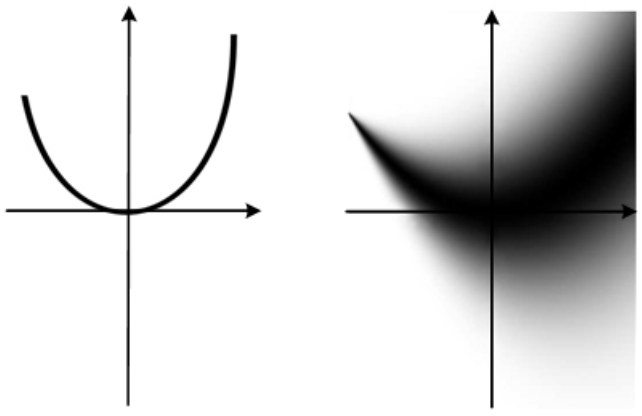


Figure 2.17: Function without noise (left) and with noise (right).

The problem of noise is that it is not possible to say for certain which of two solutions is the better one (Tan and Goh, 2008). Noise may cause two types of errors that influence the evolutionary selection negatively:

Type 1 : An inferior solution is erroneously believed to be superior and therefore survives and is given the opportunity to reproduce, or

Type 2 : A superior solution is erroneously believed to be inferior and is therefore eliminated.

In the following, an example to illustrate this problem in the context of the aforementioned production system is presented. Consider the two solutions A and B presented in Table 2.1. The throughput and average lead time of both solutions are sampled twice. Due to noise, the measured values are randomly perturbed. Depending on which specific sample is considered, the dominance relation between A and B is different. If, for example, sample #1 of A is compared to sample #1 of B, A has the highest throughput and the lowest average lead time and is therefore the best solution. However, if instead sample #2 of A is compared with sample #2 of B, the latter is the best one.

Noisy objective values like these are likely to cause a reduced convergence rate of the optimisation and a deterioration of the quality of the final sub-optimum (Beyer, 2000; Arnold and Beyer, 2002; Branke and Schmidt, 2003; Jin and Branke, 2005), since the evolutionary process, more or less, degenerates into a random search (Tan and Goh, 2008).

Solution	Throughput (maximise)		Average lead time (minimise)	
	Sample #1	Sample #2	Sample #1	Sample #2
A	23	20	31	34
B	22	25	33	28

Table 2.1: Unclear dominance relationship.

2.4.2 Techniques to deal with noise

To deal with noise in evolutionary optimisation, three basic approaches have been proposed: explicit averaging, implicit averaging, and selection modification (Jin and Branke, 2005). In *explicit averaging*, the same solution is simulated a number of times and the objective values are averaged. Simulating a solution n times reduces the noise by a factor of $\sqrt{n}$, but at the same time increases the computational effort by a factor of n (Jin and Branke, 2005). In *implicit averaging*, the sample size is adjusted to the population size; the larger the population the smaller the sample size (Fitzpatric and Grefenstette, 1988; Miller and Goldberg, 1996). The assumption of this approach is that there are many similar solutions in a large population, and that the influence of noise is compensated for as the algorithm revisits promising regions of the search space frequently. In *selection modification*, the ranking and/or selection procedure is modified to compensate for noise, such that a solution is only considered better than another solution if certain conditions are satisfied (Jin and Branke, 2005). For example, the probability of dominance can be considered as proposed by Hughes (2001), or the closeness of solutions can be considered as proposed by Babbar et al. (2003).

2.5 Summary

This chapter presented multi-objective optimisation and the difficulty of considering several conflicting objectives simultaneously. Five of the most important multi-objective evolutionary algorithms were described; MOGA, NPGA, NSGA-II, SPEA, and PAES. The concept of parallelism was also presented as a mean to increase the efficiency of evolutionary algorithms by evaluating solutions concurrently on multiple processing nodes. Another technique to improve the efficiency of evolutionary algorithms that was discussed in the chapter was surrogate evaluation functions. A surrogate evaluation function is a computationally cheap simplification of a time-consuming simulation that can be used for guiding the optimisation. Two techniques

for constructing such functions were described; surrogate approximations (black-box approach) and surrogate models (white-box approach). At the conclusion of the chapter, the problem of noisy solution evaluations in the evolutionary process were described and various noise handling techniques were briefly outlined, including explicit averaging, implicit averaging, and selection modification.

In the following chapter, a new evolutionary algorithm that makes use of the concepts presented in this chapter, that is, multi-objective optimisation, parallelism, surrogate usage, and noise handling, is described.

Chapter 3

A new evolutionary algorithm for simulation-based optimisation of real-world problems

In order to prove the hypothesis presented in the beginning of the thesis[1], a novel evolutionary algorithm for simulation-based optimisation has been developed. This algorithm, called "multi-objective parallel surrogate-assisted evolutionary algorithm" (MOPSA-EA), adapts existing techniques and also adds new concepts in order to efficiently deal with real-world problems. The fundamental characteristics of MOPSA-EA are presented in Section 3.1 and its basic steps are described in Section 3.2. Implementation details of the algorithm are discussed in Section 3.3. To deal with surrogate imprecision and noise, two new techniques have been developed and used in the algorithm. These techniques are described in Section 3.4 and Section 3.5 (respectively).

[1] The hypothesis is: "*An evolutionary algorithm that integrates techniques for multi-objective optimisation, parallelism, surrogate usage, and noise handling can achieve improved optimisation results when undertaking simulation-based optimisation of real-world problems*" (see Section 1.2.2).

3.1 Algorithm fundamentals

The previous chapter includes a presentation of multi-objective optimisation, parallelism, surrogate usage, and noise handling. The fundamental implementation of these techniques in MOPSA-EA is described below.

Multi-objective optimisation approach The most widely used approach for multi-objective optimisation is the weighted sum method (Kim and Weck, 2006). In this method, multiple objectives are combined into a single objective function by multiplying each objective function by a weight and summing up all the weighted objective functions. In this way, the multi-objective problem is solved by means of single-objective optimisation. Although the weighted sum method is intuitive and easy to implement, it suffers from two main drawbacks (Deb, 2004). First, the solution obtained will depend on the relative weight values specified by the user and devising meaningful combinations of weights is non-trivial. Second, objectives are often represented using different measurement units, which means a proper scaling must then be found for the objectives to become equally important.

Contrary to the weighted sum method, the Pareto approach to multi-objective optimisation (described in Section 2.1.1) is parameterless and also provides the user with more information about the trade-off among the various objectives (Deb, 2004). This has motivated the use of the Pareto approach in MOPSA-EA for the handling of multiple objectives. The algorithm monitors the set of Pareto-optimal trade-off solutions in each step based on the concept of dominance (see Section 2.1.1).

Parallelisation model The fact that multi-objective algorithms search for a set of Pareto-optimal solutions, instead of a single solution, may suggest using the island model for parallelisation (described in Section 2.2.2). With the island model, the population is divided into multiple sub-populations that specialise in specific objectives, and in this way the Pareto front can be explored in an efficient way. Due to

this advantage, the island model is the most popular one in parallel multi-objective evolutionary algorithms (Coello Coello et al., 2007). However, a problem with the island model is that finding a suitable partitioning of the optimisation problem is impossible without prior knowledge about the topology of the search space. In comparison, the master-slave model is simple and does not require the user having any prior knowledge about the search space. Furthermore, the master-slave model has been considered to be especially useful for problems involving computationally expensive problems (Veldhuizen et al., 2002; Mostaghim et al., 2008). A disadvantage of the master-slave model is that it involves communication overheads due to the frequent message passing between the master node and the slave nodes. However, when optimising real-world problems that involve evaluations which take several minutes, a few milliseconds of communication time is negligible. The master-slave model has been selected for MOPSA-EA as it is simple and efficient. The diffusion model is similar to the master-slave model, but imposes constraints on the network topology of the parallel processing nodes and is therefore regarded to be inappropriate in a general-purpose algorithm like MOPSA-EA.

Most multi-objective evolutionary algorithms use a generational design, even though this is not optimal from a parallel perspective, (Chafekar et al., 2005). In these algorithms, results for a complete generation must be available in order for the search to proceed with the next generation. However, this is inefficient if the population size is not divisible by the number of processing nodes, or if evaluations of different nodes take different amounts of time. Furthermore, if the population size is less than the number of processing nodes, all computing resources will not be utilised. In comparison, a steady-state design enables a higher degree of parallelism, since new solutions are continuously created (see Section 1.1) and the number of parallel evaluations is therefore not limited by the population size. Consequently, MOPSA-EA is based on a steady-state design in combination with the asynchronous master-slave model for maximum parallelism. Contrary to the synchronous model,

the asynchronous model has virtually no idle time, which can increase the number of evaluations in a given time interval (Coello Coello et al., 2007).

Besides their parallel efficiency, steady-state algorithms are known to have a high selection pressure (i.e. strong emphasis on solutions with high fitness values) (Lozano et al., 2008). A high selection pressure results in a fast convergence, but may cause the search to stagnate in a local optimum due to loss of population diversity (Lozano et al., 2008). To avoid a premature convergence, MOPSA-EA promotes diversity by using a crowding method that favours dissimilar solutions.

Surrogate usage To reduce computational time, MOPSA-EA not only supports a high degree of parallelism but also incorporates a surrogate. The surrogate is used to screen candidate solutions and identify the most promising ones. Instead of generating only a single offspring, which is normally done in steady-state algorithms, a pool of multiple offspring is created. Each of the offspring is evaluated by the surrogate, and the best one is simulated and inserted into the population. With this approach, the surrogate can help as long as its prediction is better than a random guess.

In selecting offspring to be included in the population, the imprecision associated with the surrogate is considered by using a new approach for multi-objective optimisation. As discussed in the previous section, constructing an accurate surrogate is hard, especially in real-world problems with sparse data samples. In order to achieve successful results, the imprecision of the surrogate must be considered in the optimisation, otherwise the search can be compromised with substantial performance degradation as a consequence.

Regarding the surrogate technique, MOPSA-EA is designed to be neutral with respect to this aspect. It is therefore possible to adopt both surrogate approximations (e.g. artificial neural networks or radial basis function networks) and surrogate models in the algorithm (see Section 2.3.1). Surrogate approximations are generally easier to construct since neither knowledge of the simulation internals nor programming

skills is required. Instead, the user only specifies parameter values for the surrogate approximation and it learns to imitate the simulation by itself. However, sometimes constructing a useful surrogate approximation is not possible due to a limited number of data samples, and a surrogate model has then to be used instead.

Noise handling In order to efficiently deal with noise, a new technique for multi-objective optimisation has been developed and used in MOPSA-EA. This technique uses an iterative resampling procedure that reduces the noise until the likelihood of selecting the correct solution reaches a given confidence level. To achieve an efficient utilisation of resources, the number of samples used per solution varies based on the amount of noise in the present area of the search space.

3.2 Overall procedure of algorithm

The general procedure of MOPSA-EA is shown in Figure 3.1. First, an initial population of random solutions is created and evaluated using the simulation. In case a surrogate approximation and not a surrogate model is used, this is initiated using the simulated samples from the first population (a surrogate model is static and not modified during an optimisation, as described in Section 2.3.1). A pool of offspring solutions is created through crossover between solutions in the population. The offspring are evaluated using the surrogate, and their objective values are modified to consider the imprecision of the surrogate. The offspring with the best objective values is identified, mutated, and evaluated using the simulation. In case of a surrogate approximation, the simulation sample obtained from the evaluation is used to update the surrogate. The selected offspring replaces the worst solution in the population, and the procedure continues generating a new pool of offspring.

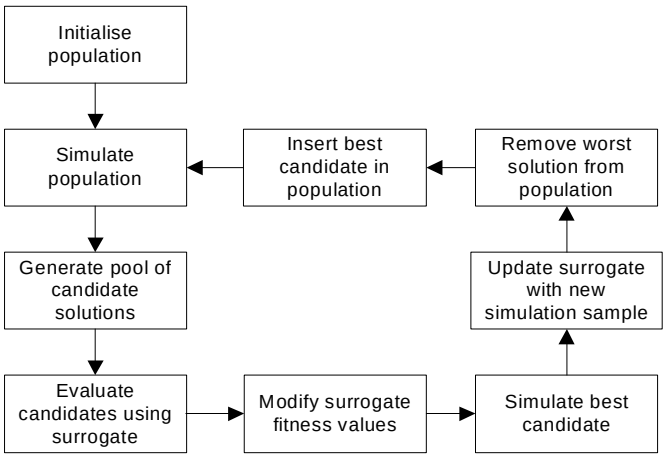


Figure 3.1: General procedure of MOPSA-EA.

In the next section, the details of the implementation of the algorithm are described.

3.3 Algorithm implementation details

The basic implementation of MOPSA-EA is described with pseudo code in Algorithm 1. Initially, the first generation of the population P is filled with random solutions. While the population is not full and there are processing nodes available, new random solutions are created by the master node and sent to the slaves for exact evaluation. When evaluated solutions are returned from the slaves, the master immediately generates new offspring to be evaluated. Offspring are created from parents in P chosen using *crowding tournament selection* (Deb, 2004). With tournament selection, solutions that have worse objective values may also be selected, which maintains diversity in the population and prevents premature convergence (see Appendix B). For the tournament, two solutions A and B are chosen randomly and A is declared the winner if either

(i) A has a better rank than B, or

(ii) A and B have the same rank, but A has a larger crowding distance than B.

Crowding distance measures the density of solutions surrounding a particular solution, and is the sum of side lengths of the hyperrectangle[2] that includes this solution without including any other solution (Deb et al., 2000), as illustrated in Figure 3.2 for a two-dimensional space.

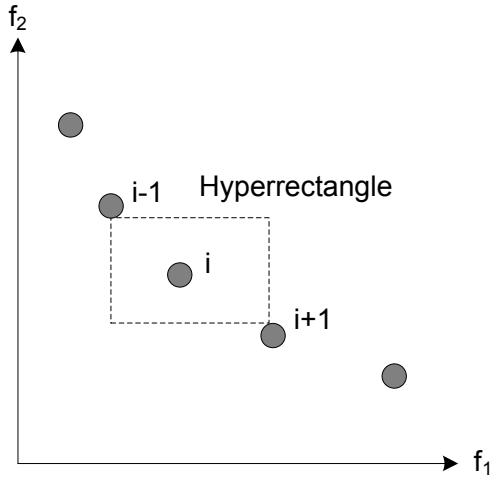


Figure 3.2: Crowding distance.

For slightly increased efficiency, the standard crowding tournament selection operator has been modified to first control if A dominates B. This way, an unnecessary non-dominated sort can be avoided[3]. The details of the optimised tournament are described in Appendix C.

When generating offspring, instead of creating only a single new solution from a pair of parents, which is a commonly used strategy in steady-state algorithms, a pool of λ candidate offspring is created. The parameter λ is usually static, but might also be self-adaptive (i.e. evolved during the optimisation) if appropriate adaption rules can be specified. An offspring pool, called O, is created as soon as a processing node becomes available. The solutions in O are evaluated by the surrogate, and since the computational cost of surrogate evaluations can be neglected in real-world optimisations (Emmerich et al., 2002), the size of the pool can be large. The surrogate objective values assigned to solutions in O are adjusted to take the imprecision of

[2] Or multi-dimensional cube, in case of more than two optimisation objectives.
[3] Although this does not affect the overall scalability of the algorithm, it reduces the cost of the critical steps of the algorithm.

the surrogate into consideration. This is done by modifying the values based on the calculated error of the surrogate (the details of this procedure are described in Section 3.4). Based on the adjusted surrogate objective values, the most promising solution in O is selected to be to inserted into P. In this procedure, all solutions of rank 1 in O are identified (called O_{R1}) and checked for domination against all solutions of rank 1 in P (called P_{R1}). By only identifying O_{R1} and P_{R1}, a full non-dominating sort is avoided. The solution in O_{R1} that dominates the most solutions in P_{R1} is selected, mutated and precisely evaluated. If several solutions in O_{R1} share the position of dominating most solutions in P_{R1}, the one having the largest Euclidean distance to its closest neighbour in P_{R1} is selected. Before the selected offspring is inserted into P, the worst solution in P is removed by performing a non-dominated sort and discarding the solution with the smallest crowding distance in the last rank. An elitistic approach is used, in which an offspring is only inserted into P if it is not dominated by any solution in P.

The sample obtained from a newly inserted offspring may be used to update the surrogate. An update need not take place every time a new sample becomes available, but can occur only every N:th sample. Less frequent updates can save a lot of time, but can also decrease the quality of the surrogate. Since the algorithm is neutral with respect to the surrogate technique, it does not specify how to update the surrogate. Surrogate update strategies vary between different techniques, and are also highly problem dependent (Jin, 2005).

Algorithm 1 Multi-Objective Parallel Surrogate-Assisted EA (MOPSA-EA).

```
function Main()
P←∅
iter←0
while (not StopOptimisation()) do
  while (ProcessingNodeAvailable()) do
    if (|P| = μ) then
      BeginSimulation(GenerateNewSolution(P))
    else
      BeginSimulation(GenerateRandomSolution())
    end if
  end while
  p←WaitForFinishedEvaluation()
  if (Mod(iter, surrogateUpdateFrequency) = 0) then
    surrogate.Update(p)
  end if
  if (|P| = μ) then
    P.Remove(SelectWorst(P))
  end if
  P.Add(p)
  iter←iter + 1
end while

function GenerateNewSolution(P)
O←∅
repeat
  parent1←SelectForReproduction(P)
  parent2←SelectForReproduction(P)
  o←Crossover(parent1, parent2)
  Mutate(o)
  SurrogateEvaluation(o)
  O←Add(o)
until (λ offspring created)
return SelectBest(O)
```

3.4 A new technique to compensate for surrogate imprecision

Surrogates are imprecise by nature and this must be considered to prevent the algorithm from being lead towards false optima. To handle the problem of surrogate imprecision in MOPSA-EA, a novel technique is presented in this section.

3.4.1 Description

In the technique proposed, surrogate objective values assigned to offspring are adjusted to consider the error of the surrogate. For each offspring, the objective errors of its parents are calculated by evaluating the parents using the surrogate and taking the difference between their assigned exact objective values and the obtained surrogate objective values. For example, if the exact values assigned to a parent are $(51,63)$ and a surrogate evaluation returns the values $(45,77)$, then the error of that parent is $(51,63) - (45,77) = (6,-14)$. Since the accuracy of the surrogate might change dynamically, surrogate values are calculated every time a solution is chosen as a parent. This means it is not the original error of a parent that is used, but the current one.

The surrogate objective values of an offspring are modified by adding the weighted mean of the error values of the offspring's parents (the error values are added since they represent the underestimation of the surrogate). The weighting is based on the similarity between the offspring and its parents, given by the crossover operator (Figure 3.3). In case only mutation is performed to create the child, the influence of the child's single parent is 100%.

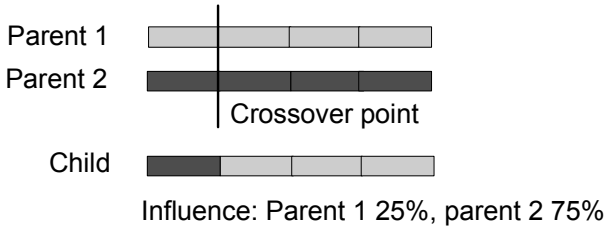


Figure 3.3: Parents' influence on child.

The adjustment of surrogate objective values can be illustrated by an example: if an offspring is assigned the values $(50,81)$ from the surrogate and the error values of its parents are $(6,-14)$ and $(8,2)$, respectively, then the new objective values of the offspring become $50+0.25*6+0.75*8 = 56.5$ and $81+0.75*-14+0.25*2 = 69$ (assuming that the parents' influence is 25% and 75%, respectively).

By adjusting the surrogate objective values according to the error of the surrogate,

the offspring selection procedure will automatically adapt to the quality of the surrogate. The larger the error of the surrogate, the more randomness in the offspring selection and the less the risk of the search being misled by the surrogate. In the same way, the smaller the error of the surrogate, the more the surrogate will impact selections.

3.4.2 Assumptions

The proposed technique of deriving a surrogate error from an offspring's parents is based on two assumptions. The first is that offspring and parents share similarities with respect to surrogate error through their genetic similarity. It has previously been shown that the objective values of an offspring can be approximated from its parents' objective values based on this assumption; a technique called *fitness inheritance* (originally described by Smith et al. 1995). Fitness inheritance has been evaluated in several studies and has been shown to work well for various problems, also in multi-objective optimisation (see for example Chen et al. 2002; Bui et al. 2005; Reyes-Sierra and Coello Coello 2005; Pilato et al. 2007). The technique of deriving surrogate errors is based on the same theory as fitness inheritance, except that a different quantification measure than fitness is inherited.

The second assumption of the proposed technique is that the error landscape of the surrogate's output is smooth. Smoothness in the error landscape means there is little difference in surrogate error between two similar outputs, while a a very rugged landscape implies a random error. Since the surrogate approximates a continuous function and computes a continuous output, smoothness can generally be assumed.

3.4.3 Discussion

For surrogate techniques based on the interpolation of points, such as Kriging, the error of the surrogate does not need to be derived as specified in the proposed technique. Interpolation techniques provide a confidence value along with an estimation, which

can be used directly to adjust the offspring surrogate values. Deriving the error of an offspring from its parents is therefore not needed, nor is it possible since the error of the surrogate at a point which has already been simulated will always be zero.

3.5 A new technique to deal with noise

When the optimisation problem is subject to noise, this must be compensated for by performing multiple samplings (i.e. simulations) of solutions, otherwise the evolutionary selection process may become unstable. Given s samplings of each solution and a total of n simulations, n/s unique solutions can be evaluated. With 500 simulations, for example, 500 different solutions can be evaluated if each of them is sampled once, and 50 if they are sampled ten times. A crucial aspect here is to find the best trade-off between the number of unique solutions evaluated and the number of samplings of each solution. The larger the number of unique solutions evaluated, the more the search space can be explored and the greater the probability of finding its optimum. However, at the same time, resamplings of solutions is necessary in order to prevent the search from being misdirected due to noise. The remainder of this section includes a description of a novel technique that efficiently addresses this trade-off problem. This technique varies the number of samples used per solution, based on the amount of noise in combination with a user-defined confidence level, controlling the trade-off between search space exploration and number of solution samplings. With the technique, resampling of solutions is performed iteratively until the noise is sufficiently reduced.

3.5.1 Basic procedure

The procedure of the proposed technique, referred to as *confidence-based dynamic resampling* (CDR), comprises five main steps which are presented below. The procedure is also illustrated in Figure 3.4 and described with pseudocode in Appendix D.

Step 1: Initial sampling

Initially, the two solutions being compared are sampled n times each to form an initial estimate of the amount of noise[4]. In order to avoid spending expensive simulations on inferior solutions, the default value of n is two. In very noisy problems, however, it might be necessary to increase the value of n.

Step 2: Calculation of mean and sample standard deviation

Based on the collected samples, the mean μ_i and sample standard deviation s_i of each objective i are calculated for the two solutions. μ_i is calculated according to Equation 3.1:

$$\mu_i = \frac{\sum_{j=1}^{N} v_j}{N}, \tag{3.1}$$

where N is the number of samples, and $v_1, ..., v_N$ are the sample values. The *sample standard deviation* measures the variability in samples (i.e the amount of noise) and is calculated according to Equation 3.2:

$$s_i = \sqrt{\frac{1}{N-1} \sum_{j=1}^{N} \left(v_j - \mu_i\right)^2}. \tag{3.2}$$

Step 3: Selection of confidence level

The confidence level is defined by the user and represents the desired certainty of the relation between two solutions. Three types of relations are possible between two solutions A and B:

(i) A dominates B, or

(ii) B dominates A, or

(iii) A and B are mutually non-dominating.

[4] It can be noted that this step is passed over if the solutions for some reason already have been sampled n times.

The confidence level α is in the interval [0,1] and specifies that in at least α of the cases the selection between two solutions should be correct (i.e. the selection should result in the same solution as if the noise would have been completely reduced). It is natural to think that the user would always want 100% of the selections to be correct, i.e. set α to 1. However, to reach a high α, a large number of solution samplings are necessary and not as many unique solutions can be evaluated. With a low α, on the other hand, many unique solutions can be evaluated, but there is a risk that the noise will not be sufficiently reduced.

A specific confidence level is defined by the user for each Pareto rank, and generally a higher rank implies a higher confidence level since high precision is usually more important for solutions in, or nearby, the Pareto front. The confidence level to use in a comparison of two solutions is derived from the solution of highest rank, which means that a non-dominating sort must first be performed to establish the ranks of the solutions. Although this sort may not return the true ranks – the goal of the procedure itself is to find out this ranking – it gives an indication of the relations between solutions in the population.

Step 4: Confidence test

In a noisy context, the true relation between two solutions can only be determined by taking the mean of all possible samples of the solutions. In practice, it is not possible to collect the complete set of samples, only a limited number of samplings can be performed. Instead, the probability that the computed relation is the same when given the collected (observed) samples as given all samples has to be established (i.e. the probability of making a correct selection from the solutions). The method of *Welch confidence interval* (WCI) is used to do this. WCI can be used to compare whether or not there is a significant difference between two solutions of unknown and possibly unequal variances with respect to a given confidence level. WCI values are calculated for each objective i according to Equation 3.3 (Law and Kelton, 2000)

$$\mu_{iA}(N_A) - \mu_{iB}(N_B) \pm t_{\widehat{f},1-\frac{\alpha}{2M}} \sqrt{\frac{s_{iA}^2}{N_A} + \frac{s_{iB}^2}{N_B}}, \tag{3.3}$$

where A and B are the two solutions being compared, μ_i is the mean of objective i (Equation 3.1), N_p is the number of samples of solution p, t is a Student t-distribution with estimated degree of freedom $\widehat{f}$

$$\widehat{f} = \frac{\left[\frac{s_A^2}{N_A} + \frac{s_B^2}{N_B}\right]}{\frac{\left[\frac{s_A^2}{N_A}\right]^2}{N_A - 1} + \frac{\left[\frac{s_B^2}{N_B}\right]^2}{N_B - 1}} \tag{3.4}$$

and probability $1 - \frac{\alpha}{2M}$ (α=confidence level, M=number of objectives), and s_i^2 is the variance of objective i (Equation 3.2). In multi-objective problems, the confidence level α has to be divided by $2M$, and not by 1 as in a single-objective problem, due to the Bonferroni Inequality (Law and Kelton, 2000). This means that for a problem of two objectives, to obtain a 0.95 confidence, for example, each objective has to be compared with a confidence level of 0.975.

If the WCI resulting from Equation 3.3 does not include 0, there is a *significant difference* between the two solutions in the i:th objective. If none of the WCIs for the M objectives include 0, it means that the relation between the two solutions (see Step 3) can be established with respect to the given confidence level. The dominating solution is then returned, or the one with largest crowding distance, if the solutions are mutually non-dominating, and the procedure is terminated. Otherwise (i.e. if any of the objective's intervals covers 0) the difference between the solutions is not significant and further noise reduction is necessary in order to determine their internal relation.

Step 5: Noise reduction by resampling

Ultimately, the relation between the solutions should be established using as few resamplings as possible to save simulation resources. Therefore, the strategy adopted in this step is to resample only one of the solutions at a time. The solution having the

largest sample standard deviation in the objective with the largest interval including 0 (i.e. with the largest potential to eliminate the undesired insignificance) is the one resampled. After a new sampling of this solution, the procedure is repeated from step 2 and a new check is made if further resampling is necessary.

To prevent the resampling of two solutions that are close to each other in objective space from continuing forever, the number of samplings of a solution is limited. Similar to the specification of confidence level, the maximum number of samplings is defined by the user for each rank. A larger number of samplings is usually allowed for solutions in higher ranks where a higher precision is needed. A limited number of samplings means that if a solution in this step has already reached its allowed number, it cannot be further sampled. The other solution is then resampled instead, unless it has also reached its maximum number. In such a case the dominating solution (or the one with largest crowding distance, if the solutions are mutually non-dominating) given the obtained samples is returned and the procedure is terminated. It should be noted that if too few resamplings are allowed, there is a risk that the desired confidence level will not be reached.

Since the ranks of solutions are calculated in every iteration of the procedure, the maximum number of samplings of solutions may change between iterations if their ranks change (as may also the confidence level to use when comparing them). In this way, the resampling strategy becomes dynamic and adjusts automatically to changes in the population.

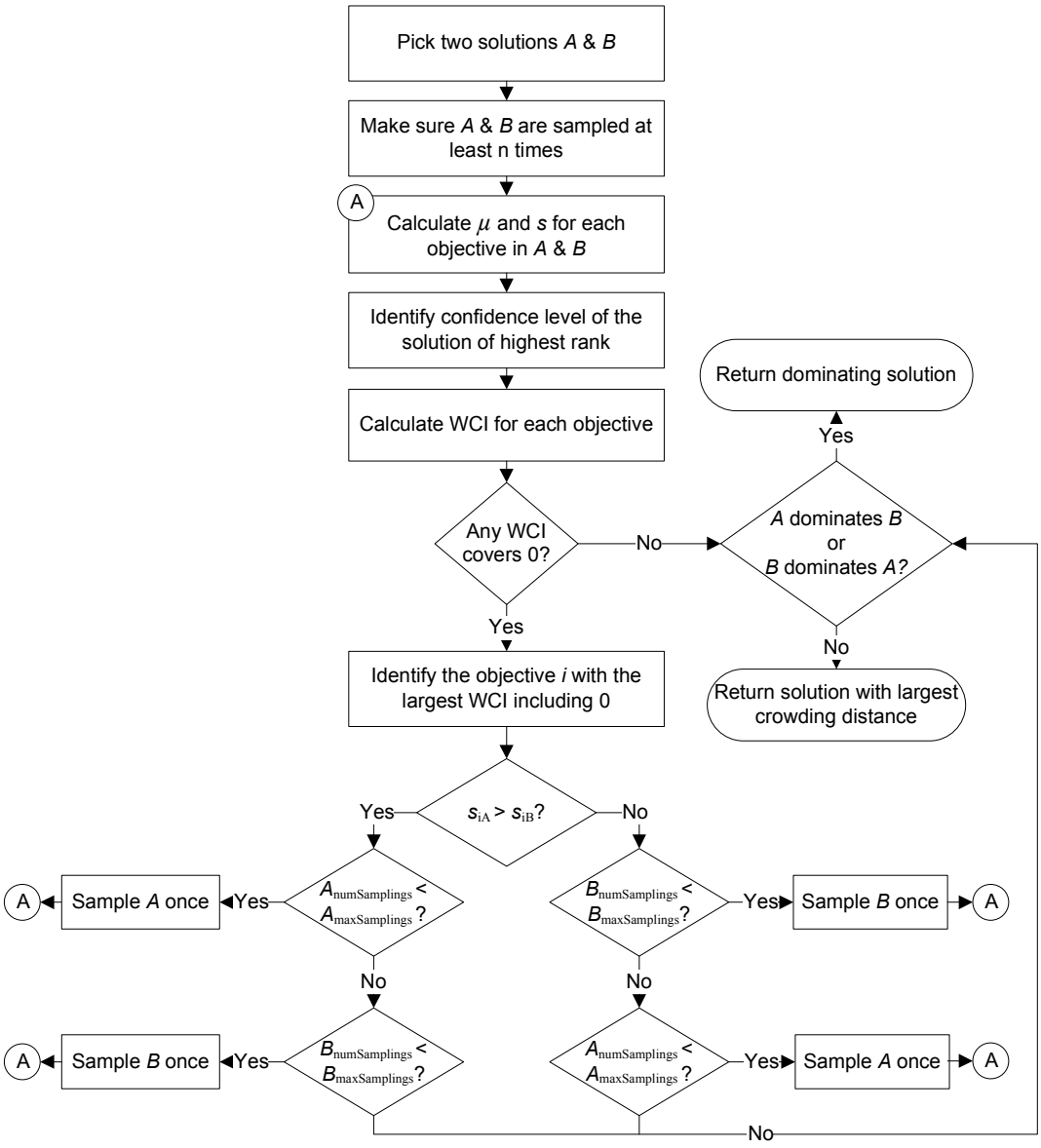


Figure 3.4: Overall procedure of confidence-based dynamic resampling technique.

3.5.2 Optimised implementation

In step 3, a non-dominating sort is performed to derive the ranks of the solutions. This is a relatively expensive sorting algorithm with a complexity of $O(MN^2)$ (where M is the number of objectives and N is the number of solutions in the population)[5]. To improve efficiency, step 3 can be optimised to avoid the non-dominating sort

[5] It can be noted that Jensen (2003) has suggested an improved version of the non-dominated sorting that reduces the complexity to $O(N \log^M N)$.

whenever possible. This is done by first checking if there is a significant difference between the two solutions in all objectives with respect to the confidence level of rank 1 (i.e. no WCI includes 0). If this is the case, there will be a significant difference between them with respect to all other confidence levels also, and their ranks do not need to be established, hence a non-dominating sort can be avoided.

Another potential optimisation of the implementation of the noise handling technique is to perform less frequent resamplings in the beginning of the search during the rough exploration, and increase the resampling frequency when the optimisation begins to converge. For example, the resampling frequency can be increased when the population has not changed in the last i iterations, or when the difference in objective values of solutions is smaller than a parameter p.

3.5.3 Discussion

The benefits of a dynamic resampling strategy have previously been discussed in Pietro et al. (2004). Pietro et al. suggest two dynamic strategies to reduce noise. In the first strategy, called *standard error dynamic resampling*, all solutions are resampled until the standard error of the mean for each solution is below a user-defined threshold (the same threshold is used for all solutions). In the second strategy, called *m-level resampling*, different thresholds are used for different noise intervals. Although Pietro et al. was able to demonstrate the potential of a dynamic procedure, they also found that properly applying a dynamic resampling strategy requires a more sophisticated approach than simply forcing the standard error below some threshold. The CDR technique is a step in this direction; instead of using standard error thresholds, statistical tests based on the concept of Welch confidence interval are being used. Furthermore, the resampling procedure used in the CDR technique is more efficient since noise is not reduced for *all* solutions in the population, but only for those participating in the evolutionary selection. The motivation of this approach is that noise is not harmful to every element of an evolutionary algorithm, but only to those

evolutionary processes that involve comparative selection. In other evolutionary operations, such as mating or mutation, noise is irrelevant.

3.6 Summary

This chapter has presented the "multi-objective parallel surrogate-assisted evolutionary algorithm" (MOPSA-EA), a new evolutionary algorithm for simulation-based optimisation of real-world problems. To reduce the time-consumption of the optimisation process, MOPSA-EA implements the master-slave parallelisation scheme in combination with a steady-state design. The algorithm also employs a surrogate for improved efficiency. The surrogate is used to identify which of the most promising offspring to include in the population from a pool of candidate solutions. In the selection of the offspring, the imprecision associated with the surrogate is compensated for using a novel technique for multi-objective optimisation. In this technique, the surrogate objective values of an offspring are modified by adding the weighted mean of the surrogate error values of the offspring's parents. In order to handle the problem of noise, the algorithm uses a new technique that effectively reduces noise through a dynamic resampling procedure.

In the following chapter, MOPSA-EA is discussed in the context of similar algorithms and techniques.

Chapter 4

Comparison with other approaches

This chapter presents a comparison of MOPSA-EA with existing approaches. In Section 4.1, evolutionary algorithms similar to MOPSA-EA are discussed. Section 4.2 describes techniques for handling surrogate imprecision, while Section 4.3 focuses on noise handling techniques.

4.1 Evolutionary algorithms

In this section, existing evolutionary algorithms that share basic design principles with MOPSA-EA are discussed. Basic design principles are (*i*) a Pareto approach to handle multiple optimisation objectives, (*ii*) an evolutionary process based on a steady-state principle, and (*iii*) adoption of surrogates. The first principle, that is, multi-objective Pareto optimisation, is considered a minimum requirement. This is because algorithms concentrating only on a single solution are fundamentally different from those optimising a Pareto set, and the results of the two different paradigms cannot be directly compared. Section 4.1.1 includes a description of evolutionary algorithms based on the first two design principles, while Section 4.1.2 discusses evolutionary algorithms making use of the first and the third principle. In Section 4.1.3, an evolutionary algorithm adopting all three design principles is presented.

4.1.1 Multi-objective steady-state evolutionary algorithms

In this section, three existing multi-objective evolutionary algorithms with a steady-state design are presented and compared to MOPSA-EA.

Objective switching genetic algorithm for design optimisation

Description *Objective switching genetic algorithm for design optimisation* (OSGADO) is a steady-state genetic algorithm that optimises multiple objectives in sequential order (Chafekar et al., 2003). Every objective is optimised in isolation during a period of time, then a switch occurs and the next objective is optimised. This process continues until the maximum number of evaluations is completed. When replacing solutions in the population with new ones, the fitness value of the objective currently being optimised is considered in combination with a crowding strategy. In this crowding strategy, a new solution replaces the closest solution with worse objective values in the population. The motivation for selecting the closest solution is to maintain the diversity of the population.

OSGADO is evaluated on four test problems, as well as two example problems from the engineering domain (two-bar truss design and welded beam design). All six problems have two objectives. The results of the evaluation show that the performance of the algorithm is about the same as the performance of NSGA-II.

Comparison with MOPSA-EA A main difference between OSGADO and MOPSA-EA is their approach in considering multiple objectives. OSGADO optimises the different objectives separately, while MOPSA-EA optimises all objectives at the same time. In almost all multi-objective problems, especially in the real world, all objectives must be optimised simultaneously since they are inter-dependent (Srinivas and Deb, 1995; Zitzler, 1999; Mehnen et al., 2004; Deb, 2004). If only one objective at a time is considered, convergence to individual objective optima is likely.

Another fundamental difference between OSGADO and MOPSA-EA is the replacement strategy. While OSGADO only considers the objective currently being optimised during the selection of a solution to replace in the population, MOPSA-EA considers all objectives. A problem of the strategy of OSGADO is that a solution might be poor in one objective, but have the best performance in all other objectives. If such a solution is discarded, it does not have the opportunity to pass on its good genes. In the replacement strategy, there is also a difference between OSGADO and MOPSA-EA with respect to the approach of maintaining diversity. The strategy of OSGADO aims to replace the solution most similar to the one being inserted, for the purpose of keeping population diversity unaltered. The strategy of MOPSA-EA, on the other hand, aims to replace the solution that is most similar to the other solutions in the population, for the purpose of increasing the diversity. In multi-objective optimisation, it is important to have solutions that are as diverse as possible in order to cover the complete Pareto front and present the user with as many different trade-offs as possible (Deb, 2004). Therefore, similar solutions are often disparaged (Deb, 2004), as in the strategy of MOPSA-EA.

Steady-state ϵ-multi-objective evolutionary algorithm

Description The *steady-state ϵ-multi-objective evolutionary algorithm* (SS-ϵ-MOEA) is a steady-state variant of PAES (for a description of PAES, see Section 2.1.2) (Deb et al., 2003). Like PAES, there are two co-evolving populations; an algorithm population and an archive population. For mating, one solution from each population is chosen and their offspring is compared to both populations for possible inclusion. In the archive population, it is checked for inclusion based on the concept of ϵ-dominance. A solution a is said to *ϵ-dominate* another solution b with respect to objective i if

$$\epsilon_i + a_i \geq b_i, \tag{4.1}$$

where ϵ is a constant, separately defined for each objective (Laumanns et al., 2002). The search space is divided into hyperboxes and a new solution is inserted in the archive only if there is not already a solution in its hyperbox, in order to maintain diversity in the archive. For inclusion in the algorithm population, the ordinary dominance concept is used. An offspring is inserted into the algorithm population if it dominates, or is mutually non-dominating to, any of the population members. If it dominates any of the solutions, it replaces one of these chosen randomly, and if it is non-dominating it replaces a solution chosen randomly from the complete population.

In the evaluation of the algorithm, four test functions with two objectives, six test functions with three objectives, and one test function with four objectives are used. Comparisons are made against NSGA-II, SPEA2, and PAES, and the results show that the performance of SS-ϵ-MOEA is better than the other algorithms on most of the test functions. In Emmerich and Naujoks (2004), the algorithm is also evaluated on a test problem of airfoil design. In this study, it is shown that SS-ϵ-MOEA can achieve slightly better results if used with a metamodel.

Comparison with MOPSA-EA While SS-ϵ-MOEA maintains two populations, MOPSA-EA has a simpler design and only maintains one. The algorithms also use different replacement strategies. SS-ϵ-MOEA uses the ϵ-dominance replacement strategy for the archive population. This requires that the user specifies an appropriate ϵ value for each objective, which might be difficult without any prior knowledge of the search space. For replacement, SS-ϵ-MOEA uses the hyperbox concept to maintain an even spread among solutions in the archive. This allows a direct control of diversity, but it is generally hard to specify an appropriate size of the hyperboxes, especially since the size of the volume in the search space in which solutions are scattered changes during the search (Deb, 2004). The replacement strategy used in SS-ϵ-MOEA for the algorithm population is considerably less complex; in case of non-domination, a random solution in the population is simply chosen for replacement. With a random

strategy, the convergence rate may be reduced since good solutions are replaced just as likely as bad ones. Furthermore, this may result in poor diversity, since diversity among solutions is not considered.

Both replacement strategies used in SS-ϵ-MOEA differ from the strategy used in MOPSA-EA. In MOPSA-EA, the solution in the worst rank of the most crowded area is replaced. This eliminates poor solutions and improves diversity, and does not require user-defined parameters.

Simple evolutionary algorithm for multi-objective optimisation

Description *Simple evolutionary algorithm for multi-objective optimisation* (SEAMO) is a standard steady-state evolutionary algorithm except that the selection of parents is not based on objective values (Mumford, 2004). Instead, every solution in the population is sequentially selected to serve as the first parent once, and paired with a second parent that is selected randomly. The offspring created from the parents is included in the population by considering the following rules (in order):

1. Are any of the individual objective values of the offspring better than the current best value of that objective? In that case, the offspring replaces one of its parents.

2. Does the offspring dominate any of its parents? In that case, it replaces the dominated parent.

3. Is the offspring non-dominating to both its parents? In that case, it replaces a solution that it dominates chosen randomly in the population.

4. Otherwise, the offspring is discarded.

SEAMO is evaluated on five test functions with two objectives and compared to PAES, NSGA-II, and SPEA2. On average, the performance of SEAMO is similar to the other algorithms.

Comparison with MOPSA-EA The approach to determine if a new solution should be included in the population and which solution it should replace is a main difference between SEAMO and MOPSA-EA. In SEAMO, the new solution is only compared to its parents and parents are more likely to be replaced. In MOPSA-EA, comparison and replacement consider all population members. Several problems with the strategy used in SEAMO are noticeable. First, in step 1, objectives are considered in isolation and convergence to individual objective optima is therefore likely. Second, in steps 1 and 2, even if a parent competes well in the population, it may be replaced (instead of replacing a poor solution), which might slow down convergence. Third, an offspring will, on average, have 50 % genetic material in common with each of its parents, and replacing an offspring with a parent therefore easily gives rise to poor diversity in the population. Fourth, in step 4, the offspring is discarded if it is not better than its parents, although it might be better than many of the population members.

General remarks on multi-objective steady-state evolutionary algorithms

Parallel solution evaluations are a relatively easy way of increasing the efficiency of the optimisation process and steady-state algorithms enable a high degree of parallelism. Nevertheless, none of the discussed multi-objective steady-state evolutionary algorithms considers parallelism. Another aspect that is not considered in any of the evolutionary algorithms is the applicability of the algorithms on real-world problems; evaluations are done on theoretical problems exclusively.

4.1.2 Surrogate-assisted multi-objective evolutionary algorithms

This section presents five multi-objective evolutionary algorithms adopting surrogates and compares them to MOPSA-EA.

Approximate pre-evaluation genetic algorithm

Description In *approximate pre-evaluation genetic algorithm* (APE-GA), the population is first evolved for a user-defined number of generations using the simulation, and the resulting Pareto front is stored in an archive (Giotis et al., 2000). In subsequent generations, the population is first pre-evaluated using local surrogates in the form of radial basis function networks. Separate surrogates are built for each solution to be evaluated using data samples from the ten simulated solutions closest in input space to the solution for which the surrogate is built. A user-defined percentage of the best solutions according to the surrogate is simulated and checked against the Pareto archive for inclusion; a solution is included if it belongs to the Pareto front kept in the archive. The mating of solutions takes place from the combined set of the population and the archive.

The evaluation of APE-GA is done on a theoretical problem from the field of airfoil design. The aim is to design an airfoil that yields given pressure distributions at two operating points. The algorithm is tested with and without using surrogates. Results from the evaluation show that the surrogate-assisted variant provides a Pareto front that has less spread, but a slightly larger number of solutions.

Comparison with MOPSA-EA A main difference between APE-GA and MOPSA-EA is how the surrogate is used. In APE-GA, the surrogate is used to evaluate population members, while in MOPSA-EA it is used to evaluate offspring candidates. Another difference is in the number of solutions simulated in each iteration. In APE-GA several solutions are simulated in each iteration (the best population members), while in MOPSA-EA only a single solution is simulated (the best offspring). In APE-GA the simulated solutions are treated interchangeably with solutions evaluated only by the surrogate when performing evolutionary operations. Due to the inherent difference between the simulation and a surrogate, their objective values cannot be directly compared and mixing them can lead to poor convergence. MOPSA-EA does not

suffer from this problem, since objective values obtained from the simulation and the surrogate are used separately.

Inexact pre-evaluation multi-objective evolutionary algorithm

Description In *inexact pre-evaluation multi-objective evolutionary algorithm* (IPE-MOEA), all population members are first evaluated using local radial basis function networks and solutions of rank 1 are then simulated (Karakasis and Giannakoglou, 2004, 2005). The number of simulations allowed in each iteration is limited, which means that it might not be possible to simulate all solutions of rank 1. In the subsequent step, the next generation of the population is formed by indiscriminately treating surrogate objective values and simulation objective values in the algorithm's selection operator. The algorithm then continues evaluating the new population with the local radial basis function networks.

IPE-MOEA is evaluated using a test function and a theoretical problem from the field of turbomachinery, both with two objectives. Results from the test function show that the algorithm is superior to an evolutionary algorithm not using surrogates, while their performance is about the same in the turbomachinery problem.

Comparison with MOPSA-EA IPE-MOEA is similar to APE-GA to a great extent, and has the same differences when compared to MOPSA-EA. A minor disparity between APE-GA and IPE-MOEA is in the selection of solutions to simulate. In APE-GA, a percentage share of the best solutions are simulated, while in IPE-MOEA all solutions of rank 1 are simulated, or as many of these that can be afforded. A disadvantage of this approach is that the user needs to specify a rule for which solutions to choose if the number of solutions of rank 1 exceeds the allowed number of simulations.

Surrogate-assisted NSGA-II

Description *Surrogate-assisted NSGA-II* (SS-NSGA-II) is a variant of NSGA-II (for a description of NSGA-II, see Section 2.1.2) (Voutchkov and Keane, 2006). In SS-NSGA-II, 20 simulations are initially carried out to build a Kriging surrogate. The optimisation is then started and evaluations of the population are performed with the surrogate for 50 generations. The population consists of 50 solutions, of which 20 are selected to be simulated. For simulation, evenly-spaced (based on Euclidean distance) solutions from the Pareto front in objective space as well as in input space are selected. A Pareto archive of simulated solutions is maintained and a newly simulated solution is included in this archive if it is not dominated by any solution in the archive. After simulations have been carried out, the population is optimised again using the surrogate for 50 generations. This procedure is repeated 20 times.

SS-NSGA-II is evaluated on three test functions with two objectives, and comparisons are made between various surrogate approximations (including Kriging models and radial basis function networks). The analysis of the evaluation shows that no surrogate technique is clearly superior to any other, but the results vary depending on the features of the function optimised. The performance of SA-NSGA-II is not compared to any other algorithm.

Comparison with MOPSA-EA In SS-NSGA-II, the optimisation is performed without using the simulation for 50 succeeding iterations. This is in contrast to MOPSA-EA, in which the simulation is used in every iteration. By only performing exact evaluations using the simulation every 50th generation as in SS-NSGA-II, the computational expense can be greatly reduced, but the optimisation will most likely converge towards local optima of the surrogate. Another difference between SS-NSGA-II and MOPSA-EA is in selecting which solutions to simulate. In MOPSA-EA, solutions are selected based on their surrogate objective values, while in SS-NSGA-II a share of the solutions are selected considering only their input parameter values. A problem of the SS-NSGA-II

approach is that there is a greater risk that simulations are wasted on inferior solutions, since performance indications are not considered. Another issue with SS-NSGA-II is that the algorithm suffers from the problem of mixing objective values obtained from the surrogate and the simulation in the evolutionary operators, as previously discussed.

NSGA-II artificial neural network

Description *NSGA-II artificial neural network* (NSGA-II-ANN) works like the standard NSGA-II, the only difference being that the simulation and an artificial neural network surrogate is used alternately to evaluate generations (Nain and Deb, 2005). In every cycle of m generations, the simulation is used to evaluate n generations, and the surrogate is used to evaluate the remaining m-n generations (see Figure 2.11). At the end of the n simulations, a new surrogate is constructed from the simulated samples obtained during the cycle. The idea of building a new surrogate in each cycle is to have local surrogates defined over a small search region.

NSGA-II-ANN is evaluated on two test functions with two objectives. The results from the experiments show that values of m and n corresponding to 10 and 3, respectively, yield the best performance. This means that in every cycle of 10 generations, the simulation is used to evaluate 3 generations and the surrogate to evaluate the remaining ones. The evaluation results also demonstrate that NSGA-II-ANN achieves better results compared to a standard NSGA-II without a surrogate in most optimisation runs.

Comparison with MOPSA-EA There is a fundamental difference between NSGA-II-ANN and MOPSA-EA in that the former is based on the concept of generational control (i.e. the simulation is used to evaluate all solutions for a number of succeeding generations), while the latter is based on individual control (i.e. in each iteration a number of solutions is simulated) (for a further description of these concepts,

see Section 2.3.1). In NSGA-II-ANN, the parameters m and n must be set carefully, otherwise the search can easily drift away towards local optima of the surrogate (Jin and Branke, 2005). Furthermore, using a fixed evolution control frequency is not very efficient since the accuracy of the surrogate may fluctuate considerably during the optimisation process (Jin and Branke, 2005).

Metamodel-assisted evolution strategy

Description *Metamodel-assisted evolution strategy* (MAES) is a surrogate-assisted version of NSGA-II (Emmerich et al., 2006). In MAES, n solutions are generated from the population of N parents, and evaluated using local Kriging surrogates. Assuming that the objectives are independent, separate surrogates are created for the different objectives. The best m from the n solutions based on rank and crowding distance are simulated. Only the simulated solutions are considered for inclusion in the population.

In the evaluation of MAES, n is set to 100, N to 20, and m to 20. Performance comparisons are made against an ordinary NSGA-II on a two-objective test problem and a theoretical airfoil problem with three objectives. For both problems, MAES achieves the best results. Three different methods for handling surrogate imprecision are also evaluated, all of them making use of the uncertainty measure provided by the Kriging surrogate along with a prediction. These methods are further discussed in Section 4.2.

Comparison with MOPSA-EA The idea of pre-screening solutions before performing simulations used in MAES is similar to the approach used in MOPSA-EA for identifying promising offspring. A main difference between the two algorithms is in the construction of the surrogates. In MAES, independent objectives are assumed and separate surrogates for the different objectives are created, while in MOPSA-EA one surrogate considers all objectives. A problem of the approach used in MAES is that objectives are seldom independent, but affect each other (Deb, 2004).

General remarks on surrogate-assisted multi-objective evolutionary algorithms

As previously discussed in Section 2.3.1, the surrogate may misdirect the search towards false optima if its imprecision is not considered. Although this is a known problem, only one of the existing surrogate-assisted multi-objective evolutionary algorithms attempts to deal with this issue (MAES), and so does MOPSA-EA. Another issue that is not considered in any of the five algorithms discussed in this section is the performance on real-world problems. Similar to the multi-objective steady-state evolutionary algorithms discussed in the preceding section, the algorithms are evaluated only on theoretical problems.

4.1.3 Multi-objective surrogate-assisted steady-state evolutionary algorithm

This section describes an evolutionary algorithm that is based on steady-state design and also uses surrogates.

Metamodel-assisted S-metric selection evolutionary multi-objective algorithm

Description *Metamodel-assisted* S*-metric selection evolutionary multi-objective algorithm* (SMS-EMOA) is a steady-state version of MAES (described in Section 4.1.2) (Emmerich et al., 2005). In SMS-EMOA, the hypervolume measure S is used in offspring and replacement selection. With this approach, the solution contributing most (or least, in case of replacement selection) to the volume in objective space dominated by obtained solutions, that is, the *hypervolume* (Figure 4.1), is selected[1]. A parent is chosen randomly from the population and n offspring are created by mutating this parent. The offspring are evaluated using local Kriging surrogates; one surrogate is constructed for each of the offspring using the $2d$ closest simulated solutions (where d is the dimension of the search space). Surrogate imprecision is

[1] Similar algorithms aiming at maximising the hypervolume have also been proposed by Igel et al. (2007), Zitzler et al. (2007), and Bradstreet et al. (2008), amongst others.

considered in the sense that solutions whose surrogate objective values are subject to a high degree of uncertainty are valued upwards to increase their chance of being simulated and thereby reduce the uncertainty. This requires a Kriging surrogate which provides an uncertainty measure along with its predictions. The technique to consider surrogate imprecision used in SMS-EMOA is further discussed in Section 4.2.1.

Evaluation of the algorithm takes place on a two-objective test problem from the domain of airfoil design and comparisons are made against the same algorithm not using surrogates. The evaluation shows that the surrogate-assisted algorithm achieves the best results.

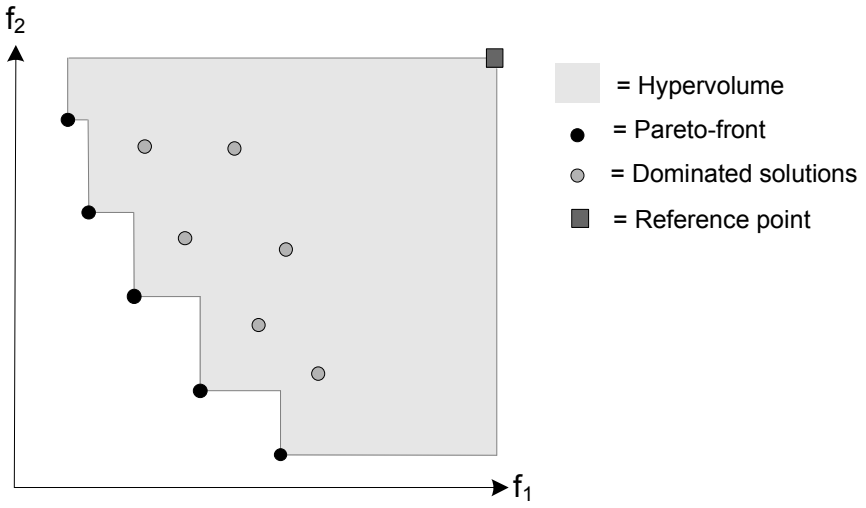


Figure 4.1: Hypervolume.

Comparison with MOPSA-EA SMS-EMOA is the existing evolutionary algorithm most similar to MOPSA-EA due to its steady-state design and its adoption of surrogates to pre-screen candidate solutions. There are, however, some differences between the two algorithms. An important difference is that SMS-EMOA is based on an evolution strategy, which means that parents are selected randomly and offspring are created only by mutation. MOPSA-EA, on the other hand, is based on a genetic algorithm and uses a tournament procedure for parental selection, and crossover in combination with mutation for the creation of offspring. A problem with selecting parents randomly is that no selection pressure exists, that is, better individuals are not

favoured in the evolutionary process. A low selection pressure generally results in a lower convergence rate (Miller and Goldberg, 1995), especially in complex problems involving time-consuming simulations (Chafekar et al., 2003). Furthermore, creating offspring by mutation alone means that a poor exploitation of the search space is likely since the search is only performed locally (Eiben and Schippers, 1998). Another fundamental difference between SMS-EMOA and MOPSA-EA is the strategy for offspring and replacement selection. While MOPSA-EA uses rank and crowding distance in these selections, SMS-EMOA uses the hypervolume measure. Using the hypervolume requires a normalised positive objective space, and a careful selection of the reference point (Emmerich et al., 2005).

4.2 Techniques to compensate for surrogate imprecision

There are three main techniques for handling surrogate imprecision in multi-objective evolutionary algorithms: improved hypervolume, probability of improvement, and expected improvement. These techniques are presented in this section and compared to the new technique of inheriting surrogate error values used in MOPSA-EA (here referred to as "surrogate error inheritance", SEI).

4.2.1 Improved hypervolume

Description In the technique of *improved hypervolume* (IHV), from among the solutions evaluated by the surrogate the one contributing most to the hypervolume (the volume in objective space dominated by obtained solutions, see Figure 4.1) is selected (Emmerich et al., 2006). The technique is designed for surrogates that, apart from the predicted objective function value, provide a measure of confidence for their predictions. Such measures of confidence can be obtained through Kriging surrogates, but not through other common surrogate techniques such as artificial neural networks

or radial basis functions. A Kriging surrogate interpolates function values and provides the mean value and the standard deviation of a Gaussian distribution that represents the probability of different outcomes to represent an exact function evaluation (Figure 2.14). In the calculation of IHV, the standard deviation of the Gaussian distribution is considered according to Equation 4.2

$$IHV = HV\left(Q \cup \left\{\hat{y} - \alpha\sigma\right\}\right) - HV\left(Q\right), \tag{4.2}$$

where HV is the hypervolume, Q is the set of non-dominated solutions, $\hat{y}$ is the surrogate estimation, σ is the standard deviation of the estimation, and α is a weight scaling the impact of σ. With $\alpha = 0$, the most promising solution according to the surrogate is selected, whereas with $\alpha = 1$ the search is directed towards regions of the search space that have been less explored. The performance of the method is strongly dependent on the value of α, especially for high dimensional problems (Ulmer et al., 2003a; Emmerich et al., 2006). Finding an appropriate weight value might, however, be difficult. An assumption of IHV is a normalised solution space, that is, that a gain in the first objective is as important as the same gain in the second objective. If not, the improvement measure might be biased towards one of the objectives (Emmerich et al., 2006).

Comparison with SEI A main difference between IHV and SEI is that IHV is restricted to surrogates which provide a standard deviation of their estimation, while SEI does not have this restriction. Another important difference between the two techniques is that IHV requires the user to specify an appropriate value of α, while SEI is free from user-defined parameters. However, with the α parameter the search can be directed towards areas of the search space for which the surrogate provides poor predictions, thereby improving the accuracy of the surrogate for these areas. This is not possible with SEI. Another difference between the two techniques is that IHV requires a normalised solution space, while SEI does not, since each objective is

treated separately. In most practical problems, it is hard to provide a normalisation a priori, and it has been stated that IHV should be used with care when an adequate normalisation cannot be ensured (Emmerich et al., 2006).

4.2.2 Probability of improvement

Description Similar to IHV, the technique of *probability of improvement* (POI) assumes a surrogate providing a standard deviation along with an estimation (Emmerich et al., 2006). In the general formula for measuring POI, each solution n in the universe, non-dominated by the current front (i.e. either members of the current front, or superior to it) is integrated over and the probability that the output of solution p equals the output of any of these solutions is calculated, according to Equation 4.3

$$POI(p) = \int_{u \in V_{nd}} pdf_p(u)\,du, \tag{4.3}$$

where pdf is the probability density function of p, and $V_{nd} = \{u \mid u \text{ is non-dominated by the current front}\}$. The solution with the largest POI value is the one selected. The general formula provided in Equation 4.3 can be implemented by performing piecewise numerical integration over m-dimensional hyperrectangles (where m is the number of objectives) and calculating the probability of p being located within one of the hyperrectangles. A drawback is that the calculation can only be computed efficiently when there are two or three objectives (Shir et al., 2007).

Comparison with SEI Similar to IHV, POI assumes a surrogate that provides a standard deviation of an estimation, which SEI does not. In contrast to POI, SEI does not include any expensive integral computations and is cheap in terms of computational cost since the only operations performed are adding and subtracting.

4.2.3 Expected improvement

Description POI has also been combined with IHV into a third technique called *expected improvement* (EI) (Emmerich et al., 2006). EI is basically an extension of POI that includes the improvement of the hypervolume achieved by adding a new solution p, according to Equation 4.4

$$ExI(p) = \int_{u \in V_{nd}} IHV(u) pdf_p(u) du, \tag{4.4}$$

where $V_{nd} = \{u \mid u \text{ is non-dominated by the current front}\}$, and IHV is the improvement function defined in Equation 4.2. Similar to POI, the formula can be implemented by performing piecewise numerical integration over m-dimensional hyperrectangles. In contrast to POI, with EI it is not only the qualitative aspect that a solution is an improvement that is captured, but also the quantity by which it improves the hypervolume. Compared to IHV and POI, however, the computational cost of EI is significantly higher (Emmerich et al., 2006).

Comparison with SEI Since EI is a combination of IHV and POI, it has the same difference compared to SEI as IHV and POI have.

4.3 Noise handling techniques

There are four major approaches that deal with noise in multi-objective problems; static resampling, modified Pareto ranking scheme, dominance-dependent lifetime, and fitness inheritance. These techniques are described in this section and compared to the confidence-based dynamic resampling (CDR) technique used in MOPSA-EA.

4.3.1 Static resampling

Description *Static resampling*, that is, sampling the objective values of all solutions a fixed number of times and using the average values, is the most commonly used

method for handling noise. The reduction of variance in the estimated objective is proportional to the sample size; sampling an objective value n times reduces the standard deviation of the objective by a factor of $\sqrt{n}$, but at the same time increases the computational effort by a factor of n (Jin and Branke, 2005).

Comparison with CDR In comparison to CDR, the technique of static resampling is easy to implement. However, it is not very efficient considering that much computational effort is spent on inferior solutions as well as on solutions not part of the evolutionary process (i.e. never influencing the search). Furthermore, with a static strategy, the same number of simulations is performed for solutions subject to little noise as for solutions subject to very much noise. In CDR, only solutions taking part in the evolutionary selection process are resampled and the number of samplings is adjusted to the amount of noise of the solutions.

4.3.2 Modified Pareto ranking scheme

Two different approaches of modifying the original Pareto ranking scheme for handling noise have been suggested; one probability-based and one based on a clustering method.

Probability-based

Description With the *probability-based Pareto ranking scheme*, the original Pareto ranking scheme is replaced by a probabilistic ranking process that takes noise into consideration (Hughes, 2001; Teich, 2001). In this ranking process, a solution s is assigned a rank representing the sum of probabilities that each of the solutions in the population dominates s (the lower the rank, the better the solution). In assigning ranks, the probability of making a wrong selection among two solutions is quantified. When considering only one objective of two solutions, estimated with value a and value b, respectively, the probability of making a wrong decision with respect to this objective

is calculated according to Equation 4.5 (assuming a minimisation problem).

$$P(a > b) = \int_{-\infty}^{\infty} \left(pdf_a(a-x) \int_{x}^{-\infty} pdf_b(b-x)\, dx \right) \tag{4.5}$$

In this Equation 4.5, x is the point being integrated over, and pdf is a probability distribution function defined by the mean and the standard deviation of the objective value. The formula estimates the probability of b being in any position left of x, when a is located at point x. In a multi-objective minimisation problem of M objectives, the probability of making a wrong decision when choosing between two solutions A and B is calculated according to Equation 4.6

$$P(A > B) = \prod_{i=0}^{M} P(A_i > B_i), \tag{4.6}$$

and the probability of A and B being mutually non-dominating is calculated according to Equation 4.7

$$P(A \equiv B) = 1 - P(A < B) - P(A > B). \tag{4.7}$$

Based on these formulae, the probabilistic rank R of solution p with index i is then calculated according to Equation 4.8

$$R_i = \sum_{j=1}^{N} P\left(p_j > p_i\right) + \frac{1}{2} \sum_{j=1}^{N} P\left(p_j \equiv p_i\right) - 0.5. \tag{4.8}$$

It is assumed that the noise distribution is known and that the noise of all solutions comes from the same normal distribution (Bui et al., 2005). These assumptions limit the application of the technique in practice, since in real-world problems the characteristics of the noise are usually unknown and the noise level might vary in the search space (Bui et al., 2005).

A variant of the probability-based Pareto ranking scheme is described in Lee et al. (2008). In this variant, the solutions are first sorted in descending order according to

the R values and then the n solutions with the best ranks are included in the Pareto front (where n is user-defined). Two types of error probabilities are calculated; (*i*) the probability that at least one solution in the non-Pareto set (i.e. solutions that do not belong to rank 1) is non-dominated, and (*ii*) the probability that at least one solution in the Pareto set is dominated. As long as these probabilities are above a user-defined threshold, resampling of solutions continues. When both errors have decreased below the given threshold, the algorithm proceeds to generate the next generation of the population.

Comparison with CDR In the probability-based Pareto ranking scheme, the noise is not reduced since solution resampling is not part of the technique. If only one evaluation of each solution is done (i.e. resampling is not performed), the evolutionary algorithm is likely to turn into a random search (see the discussion in Section 2.4). Therefore, the probability-based Pareto ranking scheme has to be complemented by a resampling scheme for reducing the noise. In contrast, CDR includes noise reduction in its procedure. Another difference between the probability-based Pareto ranking scheme and CDR is the computational expense; while the former performs expensive integral calculations, the latter only computes an interval.

Cluster-based

Description The *cluster-based Pareto ranking scheme* is also based on a modified ranking procedure (Babbar et al., 2003). In this approach, a cluster-based Pareto front is formed by solutions of rank 1 and solutions that lie in the neighbourhood of rank 1. Two solutions A and B are considered to be neighbours if their difference in mean values in the i:th objective is less then $K\sqrt{\frac{\sigma_{iA}+\sigma_{iB}}{2}}$ (where i is user-defined, K is a user-defined neighbourhood restriction factor, and σ_{ix} is the standard deviation in the i:th objective of solution x). At the same time, the difference in any objective $m \neq i$ must be less then $\sqrt{\frac{\sigma_{mA}+\sigma_{mB}}{2}}$. Initially, the value of K is large and a large number of dominated

solutions are included in the cluster-based Pareto front. During each generation of the evolutionary algorithm, all solutions in the population are resampled n times (where n is user-defined). With a decreased standard deviation resulting from the resamplings, the K value is decreased during the optimisation. A smaller K value makes it harder for solutions to be included in the front, and thereby the front becomes increasingly reliable.

Comparison with CDR In the cluster-based Pareto ranking scheme, a fixed resampling approach is adopted. The disadvantages of such an approach have been discussed in Section 4.3.1. In contrast to CDR, the amount of noise is not considered in the cluster-based Pareto ranking scheme and it is assumed that all solutions are subject to the same amount of noise.

4.3.3 Dominance-dependent lifetime

Description In the technique of *dominance-dependent lifetime*, each solution is assigned a maximal lifetime based on the number of solutions it dominates (Büche et al., 2002). A solution dominating a large number of solutions is assigned a short lifetime, and vice verse. The purpose of this strategy is to reduce the impact of solutions that appear to be good, but whose fitness value is misleading due to noise. To enable good solutions to proceed in the evolutionary process, non-dominating solutions whose lifetimes have expired are resampled and added to the population with the new objective values.

Comparison with CDR In the technique of dominance-dependent lifetime, solutions are resampled, but instead of using the average sample values, as in CDR, only the last sampling is considered. When only one sampling is considered, the noise is not reduced and the evolutionary process more or less degenerates into a random search.

4.3.4 Fitness inheritance

Description A technique to handle noise based on the concept of *fitness inheritance* has also been suggested (Bui et al., 2005). In this approach, a child inherits the mean objective value $\bar{\mu} = \frac{\mu_{parent1}+\mu_{parent2}}{2}$ and the mean standard deviation $\bar{\sigma} = \frac{\sigma_{parent1}+\sigma_{parent2}}{2}$ from its parents. The child is evaluated once, and if the objective values fall within the confidence interval $\left(\bar{\mu} - 3 * \bar{\sigma} \leq f \leq \bar{\mu} + 3 * \bar{\sigma}\right)$ the inherited fitness is accepted. Otherwise, the child is resampled a user-defined number of times and assigned the mean value of these evaluations.

Comparison with CDR In contrast to CDR, in the technique of fitness inheritance, the noise of a solution is not considered when deciding whether or not to resample it. Instead, based on a single sampling, the deciding factor is the similarity of the solution to its parents. However, due to a high amount of noise, a single evaluation may falsely indicate a high degree of similarity. In this case, the solution will inherit its parents' fitness without resampling, and, as a consequence, have an incorrect fitness. This incorrectness will propagate to new generations of the population when the solution is selected to act as a parent. When resampling of a solution is actually performed, a static resampling scheme is used. The disadvantages of such a scheme have previously been discussed.

4.3.5 General remarks on noise handling techniques

A common drawback of all techniques discussed in this section, including CDR, is that they require user-defined parameters. Since the characteristics of the noise are unknown in real-world problems, the appropriate setting of these parameters generally becomes a hard task.

It can also be noticed that none of the techniques for handling noise described in this section have been adopted in any of the evolutionary algorithms previously discussed in Section 4.1.

4.4 Summary

This chapter presented a comparison between the proposed algorithm MOPSA-EA (described in Chapter 3) and similar approaches. Firstly, nine multi-objective evolutionary algorithms that share basic design principles with MOPSA-EA (i.e. have a steady-state design and/or use surrogates) were discussed. The technique to compensate for surrogate imprecision used in MOPSA-EA was then contrasted to three existing multi-objective techniques proposed for the same purpose. At the conclusion of the chapter, MOPSA-EA's noise handling technique was discussed in relation to five existing noise handling techniques for multi-objective problems.

The following chapter presents a description of the evaluation of MOPSA-EA on a number of optimisation problems.

Chapter 5

Evaluation

MOPSA-EA has been evaluated by applying the algorithm to a number of optimisation problems, which are described in Section 5.1. The experimental platform used to realise the optimisations is outlined in Section 5.2 and the performance metrics used in the evaluation are discussed in Section 5.3. To assess the relative performance of MOPSA-EA, the algorithm has been compared to three existing multi-objective evolutionary algorithms, which are outlined in Section 5.4. Surrogate configuration and algorithm parameter settings are presented in Section 5.5 and Section 5.6, respectively. In addition, Section 5.7 describes the evaluation of the noise handling technique used in MOPSA-EA, while Section 5.8 presents the evaluation of the algorithm's parallel performance.

5.1 Optimisation problems

The evaluation has used five benchmark problems (described in Section 5.1.1) and two complex real-world problems from the manufacturing domain (described in Section 5.1.2). All these problems involve multiple objectives, since the focus of this work is on multi-objective optimisation. It should, however, be pointed out that the algorithm, as such, is general and can be applied to single-objective problems as well.

5.1.1 Benchmark problems

In the field of evolutionary optimisation, it is common to use standardised benchmark problems to assess the relative performance of different algorithms. These functions enable the comparison and replication of experiments, and are also considerably faster to run than real-world problems. A set of guidelines for the systematic development of benchmark problems for multi-objective optimisation was first proposed in Deb (1999). Based on these guidelines, Zitzler et al. (2000) suggested six benchmark functions that have been extensively used in the literature for the analysis and comparison of multi-objective evolutionary algorithms: ZDT1, ZDT2, ZDT3, ZDT4, ZDT5, and ZDT6 (Zitzler et al., 2000)[1]. These problems have properties that are known to cause difficulties in converging to the true Pareto-optimal front and reflect characteristics of real-world problems, such as multimodality, non-separability, and high dimensionality. This has motivated the use of these functions in assessing the performance of MOPSA-EA. However, function ZDT5 has been omitted since it defines a Boolean function over binary strings, and such binary encoded solutions are seldom relevant in real-world problems. The relevant ZDT functions are described in Table 5.1.

In the study of Zitzler et al. (2000), the ZDT functions were optimised using 25,000 evaluations for each of the functions. It was shown that SPEA and NSGA-II (see Section 2.1.2) are able to converge to a near-optimal Pareto front with this number of evaluations. In other studies, approximately the same number of evaluations was also used to optimise the ZDT functions (e.g. Deb et al., 2000; Huband et al., 2003; Emmerich et al., 2005). In this study, only 5000 evaluations have been allowed to emulate a scenario in which the evaluation function is computationally expensive and the optimisation time budget is limited.

[1] It can be noted there are also other benchmark functions for multi-objective optimisation, such as the DTLZ suite defined by Deb et al. (2002) or the WFG toolkit defined by Huband et al. (2005).

Function	Number of inputs	Variable bounds	Objective functions	Optimal solutions
ZDT1	30	[0,1]	$f_1(x) = x_1$	$x_1 \in [0,1]$
			$f_2(x) = g(x)\left[1 - \sqrt{x_1 / g(x)}\right]$	$x_i = 0$
			$g(x) = 1 + 9\left(\sum_{i=2}^{n} x_i\right) / (n-1)$	$i = 2, \ldots, n$
ZDT2	30	[0,1]	$f_1(x) = x_1$	$x_1 \in [0,1]$
			$f_2(x) = g(x)\left[1 - \left(x_i / g(x)\right)^2\right]$	$x_i = 0$
			$g(x) = 1 + 9\left(\sum_{i=2}^{n} x_i\right) / (n-1)$	$i = 2, \ldots, n$
ZDT3	30	[0,1]	$f_1(x) = x_1$	$x_1 \in [0,1]$
			$f_2(x) = g(x)\left[1 - \sqrt{x_1 / g(x)} - \frac{x_1}{g(x)} \sin(10\pi x_1)\right]$	$x_i = 0$
			$g(x) = 1 + 9\left(\sum_{i=2}^{n} x_i\right) / (n-1)$	$i = 2, \ldots, n$
ZDT4	10	$x_1 \in [0,1]$ $x_i \in [-5,5]$ i=2,...,n	$f_1(x) = x_1$	$x_1 \in [0,1]$
			$f_2(x) = g(x)\left[1 - \sqrt{x_1 / g(x)}\right]$	$x_i = 0$
			$g(x) = 1 + 10(n-1) + \sum_{i=2}^{n}\left[x_i^2 - 10\cos(4\pi x_i)\right]$	$i = 2, \ldots, n$
ZDT6	10	[0,1]	$f_1(x) = 1 - \exp(-4x_i)\sin^6(4\pi x_i)$	$x_1 \in [0,1]$
			$f_2(x) = g(x)\left[1 - \left(f_1(x) / g(x)\right)^2\right]$	$x_i = 0$
			$g(x) = 1 + 9\left(\sum_{i=2}^{n} x_i\right) / (n-1)$	$i = 2, \ldots, n$

Table 5.1: ZDT benchmark functions.

The ZDT1-ZDT4 functions have received some criticism for being too dependent on only one parameter, making it relatively easy to locate the optimal Pareto front (Deb et al., 2005). In this study, however, the small number of function evaluations allowed makes it difficult to find the optima in any case.

5.1.2 Real-world problems

In order to evaluate the industrial strength of MOPSA-EA, the algorithm was applied to two complex multi-objective real-world problems from the manufacturing domain.

The first problem concerned the optimisation of a manufacturing cell for the production of aircraft and gas turbine engine components at Volvo Aero. To improve processing, the overall utilisation of the cell had to be increased and the number of overdue components minimised. The second problem concerned the optimisation of a camshaft machining line at Volvo Cars Engine. In order to maximise the throughput of the line and maintain required levels of different camshaft variants in the finished goods stock, the scheduling of the line had to be improved.

More details on both these challenging optimisation problems are provided in the remainder of this section.

Engine component manufacturing at Volvo Aero

Volvo Aero develops and manufactures high-technology components for aircraft- and gas turbine engines. An example of an engine constructed from components manufactured at Volvo Aero is shown in Figure 5.1. Today, more than 80 percent of all new commercial aircraft with more than 100 passenger capacity are equipped with engine components from Volvo Aero. Components manufactured at Volvo Aero can be found in military fighter aircraft as well, such as the F/A-18 E/F Super Hornet. As a partner of the European space program, Volvo Aero is also the primary supplier of nozzles and fuel pump turbines for the Vulcain rocket engine.

Figure 5.1: Example of engine with components manufactured at Volvo Aero.

Volvo Aero's facilities are located both in Scandinavia and in the US, and in this study the focus is on a factory located at the headquarters of the Volvo Aero

Corporation in Sweden (some photos from the factory are provided in Appendix E). A new automated manufacturing cell that processes a wide range of different engine components has recently been introduced at the factory. The cell comprises five multi-task machines and five burring stations. The operations that are performed in a machine or at a station vary for different components. Instructions and tools are automatically set up in a machine for the component that arrives, which means that several different components can easily be processed in the cell at the same time. In case two or more components arrive simultaneously at a machine or station, a priority function determines which component has precedence. In this function, a critical ratio value is calculated for each component to determine how much it is behind or ahead of schedule. The critical ratio value is derived by dividing the time to due date (i.e. scheduled completion) by the time the component is expected to be finish, according to Equation 5.1.

$$CR = \begin{array}{l} \text{if } due \geq now: \ \frac{1+due-now}{1+TRPT} \\ \text{if } due < now: \ \frac{1}{(1+now+due)*(1+TRPT)} \end{array} \tag{5.1}$$

In Equation 5.1, due is the due time of the component (i.e. deadline), now is the current time, and $TRPT$ is the theoretical total remaining processing time (i.e. the active operation time plus time for machine set ups and traveling between machines/stations, without considering possible delays in the cell). A component with a critical ratio value of 1.0 is "on schedule", while it is “behind schedule” if the value is less than 1.0 and “ahead of schedule” if the value is larger than 1.0. The component with the lowest critical ratio value has precedence in a machine.

The inflow of the cell is controlled by using fixed inter-arrival times for components. The inter-arrival time not only specifies when a component should enter the system, but also determines the component’s due time since an overall production strategy is to process no more than one component of a specific type in the cell at a time. This means that if, for example, the inter-arrival time for a component type is set at two

hours, a new component of that type is introduced in the cell every two hours with a due time of two hours from the time it was introduced. For efficient production, the inter-arrival times should be specified in a way that maximises the utilisation of the cell (objective 1) and simultaneously minimises overdue components, that is tardiness (objective 2). For high utilisation, short inter-arrival times are needed in order to obtain a high cell load and thereby avoid machine starvation. However, avoiding overdue components requires generous due times; that is long inter-arrival times. This means that the two objectives of maximal utilisation and minimal tardiness conflict with each other.

For the optimisation, a discrete-event simulation model of the manufacturing cell was built using the SIMUL8 software package[2], and a scenario with eleven different component types was specified in the simulation. A screen image of the modelled cell is presented in Figure 5.2. The simulation model has a front-end interface developed in Excel, which for the user facilitates entering input parameters into the model without the need to learn the simulation language. Validity tests indicate that the simulation model represents reality well, and that it captures the stochastic variations of the cell that occur due to unpredictable machine breakdowns. A single simulation run takes approximately 7 minutes including input and output processing. The company's optimisation time budget allowed for 400 simulations.

[2] www.simul8.com

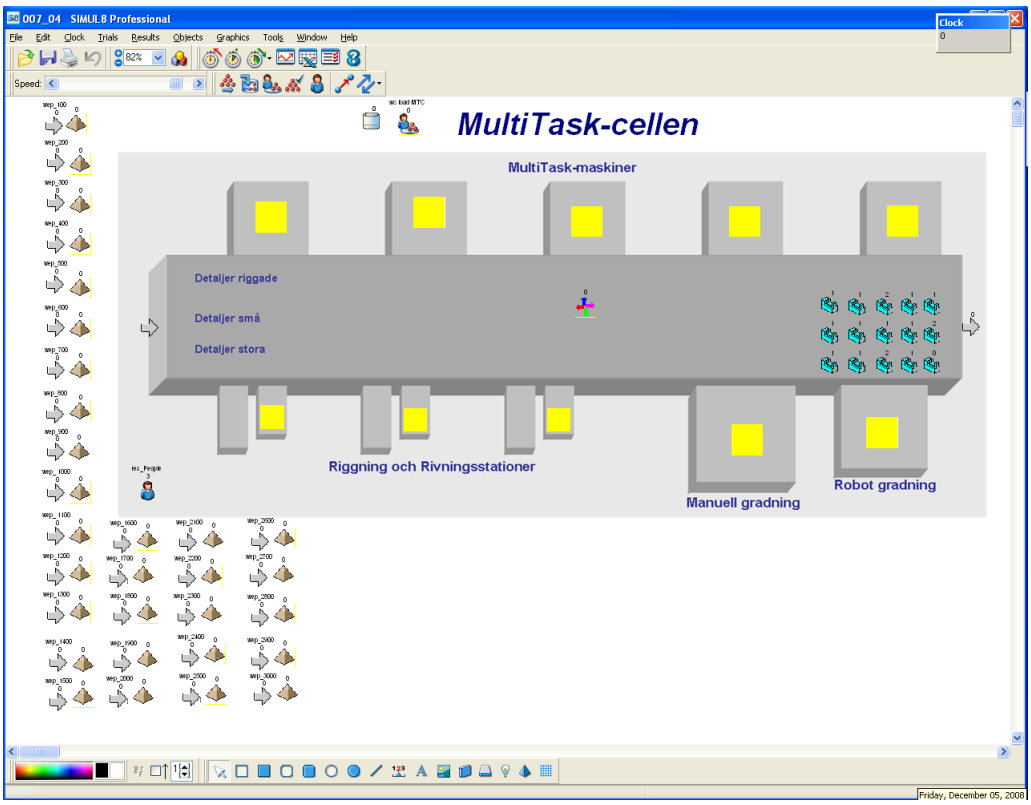


Figure 5.2: Simulation model of manufacturing cell at Volvo Aero.

The genetic representation used in the optimisation is shown in Figure 5.3. The genome consists of eleven genes, each representing a specific component type. The first gene represents component type number one, the second gene component type number two, and so on. The value of a gene is the inter-arrival time (in minutes) of the component type it represents. In the mutation procedure, there is a small probability for each gene that the value is slightly modified, i.e. that the inter-arrival time of a component is increased or decreased.

Component type 1	Component type 2	Component type 3	Component type 4	Component type 5	Component type 6	Component type 7	Component type 8	Component type 9	Component type 10	Component type 11
1383	3035	141	1064	2487	485	1852	638	188	1315	2779

Figure 5.3: Genetic representation of Volvo Aero problem.

Camshaft machining line at Volvo Cars Engine

Volvo Cars Engine manufactures passenger cars and holds a global market share of 1–2% with approximately 440,000 cars produced every year. Volvo Cars Engine's production factories are located in Sweden and Belgium, and final assembly is carried out in Asia and South Africa. In this study, a factory in Sweden responsible for manufacturing petrol and diesel engine components was selected. Appendix F includes some photos of the factory site. The specific focus of the study was on the factory's camshaft machining line, responsible for producing 15 different camshaft variants (an example of a camshaft is shown in Figure 5.4). The machining line comprises a number of operation groups, each responsible for performing a specific operation on a camshaft being produced. For each operation group, there are a number of parallel machines responsible for actually performing these operations. There are 14 operating groups and 34 machines in total. Unlike an ordinary flow shop with parallel machines, each machine has its own processing time, physical capability and limitations, as well as variability in terms of failures and set-ups. All finished camshafts are taken to a storage area, from where they can later be distributed to other production areas. It is important to always maintain stock levels in the storage area above certain limits in order to ensure that production is not delayed elsewhere because of unexpected events such as machine breakdowns or quality defects. Camshafts are produced in batches (i.e. a collection of camshafts of the same kind), with each batch being prioritised for production in order to keep stock levels in the storage area at an acceptable level. Prioritisation is determined by a schedule, which also defines the individual path through machines and operations that the batches should take. Stock levels are checked at continuous time intervals, and a mean value of the shortage noticed at each measure point is calculated at the end of the scheduling period. Besides minimising product shortage (objective 1), it is also important that a schedule results in a throughput of the line that is as high as possible for maximum efficiency . For high throughput (objective 2), there should ideally be only one variant produced in the line

at the same time to avoid set-up times for machines, which cause a large overhead. Furthermore, for a high throughput, variants with shorter processing times should be prioritised before those with longer processing times according to general scheduling theory (Blazewicz et al., 2001). However, to maintain the minimum stock levels, an even mix of the different variants being produced in the line is needed and variants should be prioritised so that shortage is avoided. In other words, the objectives of minimum shortage and maximum throughput conflict with each other.

Figure 5.4: Example of camshaft.

For the optimisation of the machining line, a discrete-event simulation was constructed using the QUEST simulation software[3] (Figure 5.5). The simulation model was primarily built for short-period scheduling, and in this study a seven day of production period was simulated in the model. Such a period includes scheduling about 500 batches (the exact number depends on customer orders for the specific period). It takes approximately 5 minutes to simulate seven days of production, input and output processing included. The company's time budget allowed for the performance of 600 simulations. The set-up for a scheduling period is defined in the model's Excel user-interface. Careful tests and analysis by domain experts at Volvo Cars Engine confirmed the validity of the model.

[3] www.3ds.com

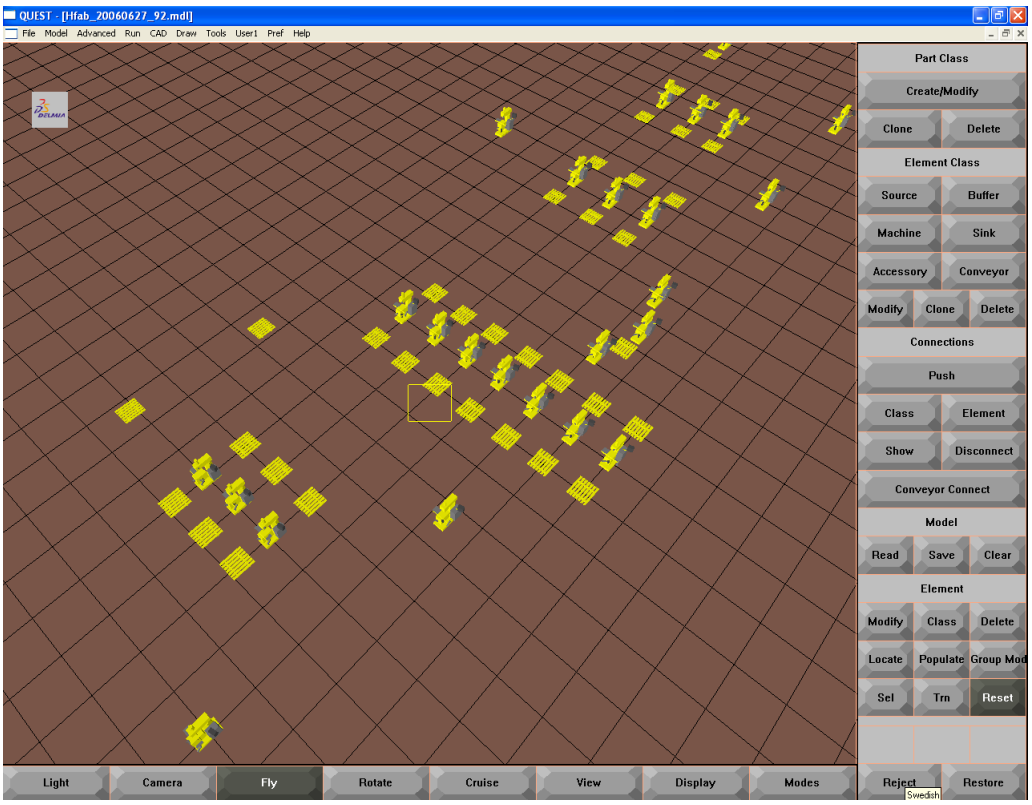


Figure 5.5: Simulation model of machining line at Volvo Cars Engine.

In the genetic representation used in the optimisation, the genome is represented as a matrix in which each row corresponds to a specific batch and each column represents a machine. Figure 5.6 shows an example of a simplified genome with six batches and seven machines. Checked cells of the matrix in the figure mark machines that are not allowed (batches in group number two can, for example, not be processed in the first, second, or fifth machine). In the example presented in the figure, M3a and M3b are parallel, equivalent machines. When batch number three arrives at these machines, it is split into two parts of 50 camshafts each to enable concurrent processing. Note that batch sizes must always be evenly divisible by 50 due to the capacity of the carts transporting camshafts in the factory.

When new solutions are created, a uniform crossover operator is used in which each cell in the matrix is taken randomly from one of the two parents. The batch sequence is simply inherited from the parents; the first child inherits its sequence from the first parent, while the second child inherits it from the second parent.

		M1	M2	M3a	M3b	M4	M5	M6
Group 1	Batch 1	50						50
	Batch 2		100			100		
Group 2	Batch 3			50	50			100
	Batch 4				100	100		
	Batch 5			50				50
Group 3	Batch 6	50					50	

Figure 5.6: Genetic representation of Volvo Cars Engine problem.

As Figure 5.6 illustrates, adjacent batches composed of the same camshaft type form a batch group. These groups are used in the mutation operator to modify the batch sequence. In the mutation procedure, a random batch group is first selected and whether this group should move up or down in the sequence is determined (the probability is 50/50). It is then decided whether the complete group should move, or just one or a few of the batches within it. Generally, it is advantageous to move several batches at a time, since moving only one batch often results in a batch mix that requires many time consuming machine set-ups. The procedure of mutating the batch sequence is exemplified in Figure 5.7. In this example, the first two batches of the second batch group are moved to the start of the sequence.

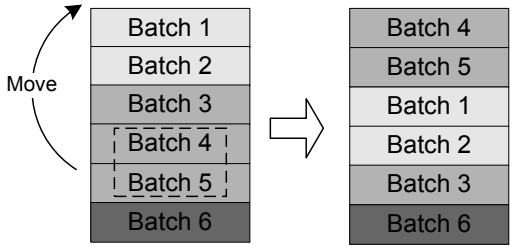


Figure 5.7: Mutation of batch sequence.

5.2 Platform

The evaluation experiments were realised using the OPTIMSE platform (Ng et al., 2007). *OPTIMISE* (OPTIMisation using Intelligent Simulation and Experimentation) is a parallel and distributed computing platform that supports multiple users in

running experiments and optimisations with different simulation systems. In the platform, various evolutionary algorithms, surrogates, stochastic simulation systems and a corresponding database management system are integrated in a parallel and distributed fashion.

The OPTIMISE platform is illustrated in Figure 5.8. In a simulation-optimisation application supported by the OPTIMISE platform, the optimisation engine is the most important component because it provides the core functionality for major algorithmic processing, and also acts as the hub for coordinating other functions (e.g. data logging and surrogate handling). Client applications can be developed to control and monitor optimisation processes and simulation experiments. Server components are accessed by the client applications through using the OPTIMISE web services, which the web server hosts. The web server listens to XML requests from clients and acts accordingly. A request may, for example, be launching/controlling a simulation-based optimisation process (through the optimisation manager), or retrieving data from the optimisation database.

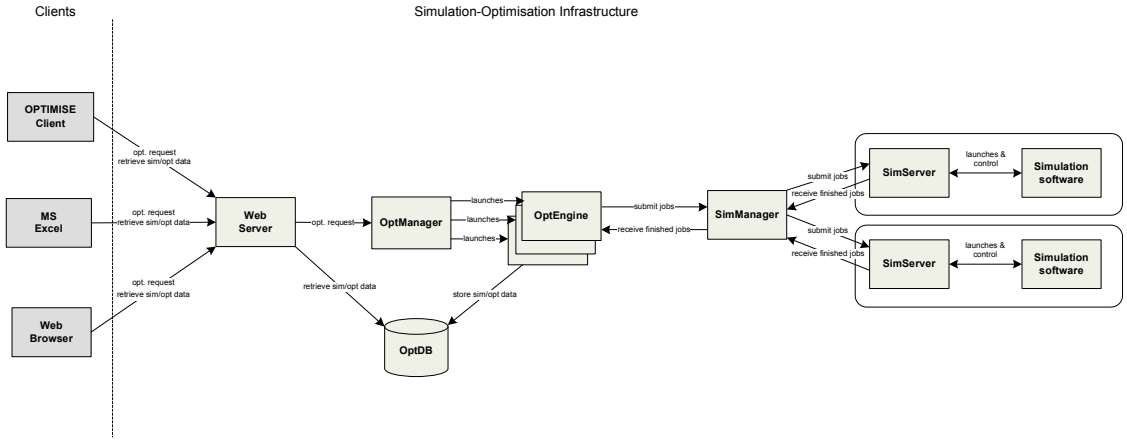


Figure 5.8: The OPTIMISE platform.

A cluster of processing nodes is associated with the OPTIMISE platform consisting of 20 identical PCs (identical with respect to both hardware and software). The characteristics of these PCs are described in Table 5.2.

Characteristic	Description
CPU	3 GHz dual core
Memory	2 Gb
Network	Gigabit Ethernet
Operating system	Windows XP Professional SP2
Communication library	MSMQ

Table 5.2: Characteristics of processing nodes in the OPTIMISE platform.

The PCs having dual-core processors means it is possible to simulate up to 40 solutions in parallel in the OPTIMISE platform.

5.3 Evaluation metrics

An overall goal in multi-objective optimisation is convergence to the Pareto-optimal front. A commonly used measure for evaluating convergence in problems having a known true optimal front (which is the case with the ZDT functions) is the Υ metric (Deb et al., 2000). This metric measures the degree of convergence by calculating the average minimum Euclidean distances from each of the obtained non-dominated solutions to the closest solution in the true Pareto front (Figure 5.9). The smaller the value of Υ, the better the convergence of the algorithm.

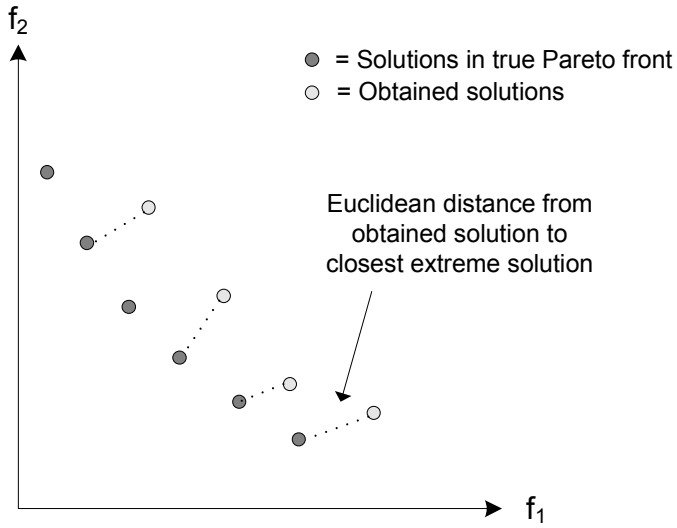


Figure 5.9: Υ performance metric.

Traditionally, the Υ metric has often been used along with the spread metric Δ, measuring the diversity among solutions in the obtained non-dominated set (for details see Deb et al. 2000). In this study it is argued that when comparing the performance of different algorithms, the Δ metric is only interesting when the number of solutions in the obtained non-dominated set is equivalent in all algorithms, which can seldom be guaranteed in practice. Therefore, the Δ metric is not used in this study. The Δ metric is also omitted in other studies, and the Υ metric is instead complemented by the S metric (also called the hypervolume metric) (e.g. Emmerich et al. 2005; Naujoks et al. 2005). The S metric is one of the most frequently applied measures for comparing the results of multi-objective evolutionary algorithms (Emmerich et al., 2005). Basically, S measures the volume in objective space dominated by obtained solutions (see Figure 4.1). The larger the volume, the better the results of the algorithm. The S metric is a combined measure of convergence and diversity in the set of non-dominated solutions. It does not assume that the true Pareto-optimal front is known and can therefore also be applied to real-world problems.

Another metric that is a combined measure of convergence and diversity is the inverted generational distance metric (Coello Coello and Reyes-Sierra, 2004). *Inverted generational distance* (IGD) aims at being a combined measure of convergence and diversity in the set of non-dominated solutions. This metric is calculated by taking the average of all Euclidean distances from each true Pareto front sample to the closest solution generated by the algorithm (Figure 5.10). The rationale behind this metric is that for a low IGD value, a well spread front and a good convergence is necessary at the same time.

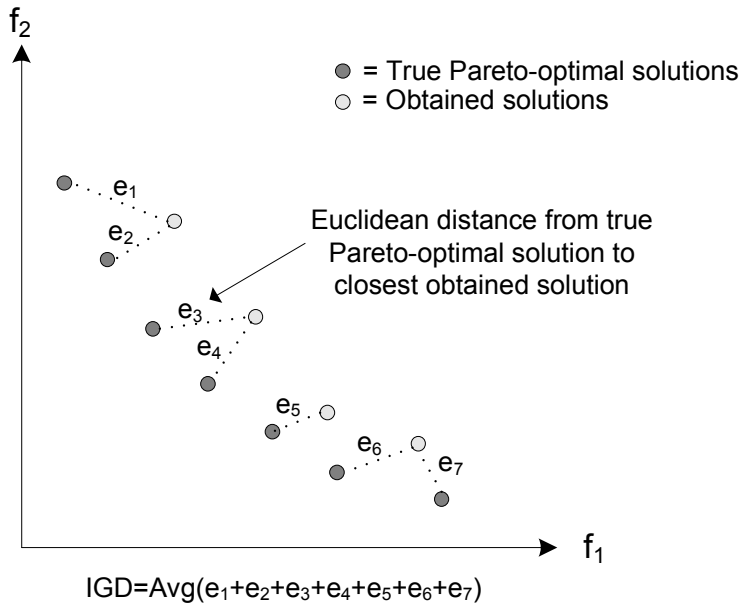


Figure 5.10: IGD performance metric.

5.4 Algorithm performance comparison

To assess the relative performance of MOPSA-EA, it must be compared to existing multi-objective evolutionary algorithms. However, only surrogate-assisted evolutionary algorithms are considered relevant for such a comparison since it has been shown that these are generally superior to algorithms that do not adopt surrogates (Jin et al., 2002). Among the set of surrogate-assisted multi-objective evolutionary algorithms presented in Section 4.1, metamodel-assisted S-metric selection evolutionary multi-objective algorithm (SMS-EMOA), metamodel-assisted evolution strategy (MAES), and NSGA-II artificial neural network (NSGA-II-ANN) were selected for comparison. SMS-EMOA and MAES were selected because their designs are the ones most similar to that of MOPSA-EA (especially since SMS-EMOA and MAES also adopt pre-screening of candidate solutions). NSGA-II-ANN, on the other hand, was selected because its design is the one most different to MOPSA-EA; NSGA-II-ANN is based on generational control instead of individual control (see Section 2.3.1 for a discussion of control concepts). In addition, an important reason for selecting SMS-EMOA, MAES, and NSGA-II-ANN is

that all these have shown particularly good performance when evaluated in previous studies.

SMS-EMOA is based on the concepts of evolutionary strategies, in which offspring are created by mutating a randomly chosen parent. This is in contrast to many other multi-objective evolutionary algorithms that are based on the concepts of genetic algorithms. Algorithms based on a genetic algorithm use a more sophisticated parental selection scheme (e.g. tournament selection), and apply both crossover and mutation when creating offspring. Experiments carried out on the ZDT functions show that the performance of SMS-EMOA improved significantly[4] when implemented with the genetic algorithm concepts, which can probably be explained by two factors:

(i) A powerful evolutionary optimisation depends on a good balance between exploitation by mutation and exploration by crossover (Eiben and Schippers, 1998), and

(ii) The convergence rate of an evolutionary algorithm heavily depends on the degree to which better individuals are favoured in the evolutionary process, called selection pressure (Miller and Goldberg, 1995). In evolution strategies, this kind of pressure is absent in the parental selection.

For a relevant and interesting performance comparison, the genetic algorithm-inspired variant of SMS-EMOA was used in the evaluation instead of the original implementation (i.e. parents are chosen using tournament selection and both crossover and mutation are applied to create offspring). To indicate that it is a modified variant of the original SMS-EMOA that is used, the algorithm is hereafter denoted “m-SMS-EMOA”. In Algorithm 2, the implementation of m-SMS-EMOA is described with pseudo code.

[4] Between 13-18%, depending on function.

Algorithm 2 Modified metamodel-Assisted S-Metric Selection Evolutionary Multi-Objective Algorithm (m-SMS-EMOA).

```
P←Create()
Simulation(P)
while (not StopOptimisation()) do
  O←∅
  parent1←TournamentSelection(P)
  parent2←TournamentSelection(P)
  repeat
    o←Crossover(parent1, parent2)
    Mutate(o)
    O←Add(o)
  until (λ offspring created)
  SurrogateEvaluation(O)
  q←SelectBest(O)
  Simulation(q)
  P.Add(q)
  P.Remove(SelectWorst(P))
end while
```

In Algorithm 3, the implementation of MAES is described with pseudo code. In the original MAES, one surrogate is created for each objective, assuming that the objectives are independent of each other. Since this assumption does not hold in most problems (see Section 2.1.1), the original MAES has been modified to use a surrogate that considers all objectives. This modified variant is hereafter denoted "m-MAES".

Algorithm 3 Modified metamodel-Assisted Evolution Strategy (m-MAES).

```
P←Create()
Simulation(P)
while (not StopOptimisation()) do
  O←∅
  repeat
    o←Crossover(TournamentSelection(P), TournamentSelection(P))
    Mutate(o)
    O←Add(o)
  until (λ offspring created)
  SurrogateEvaluation(O)
  Q←SelectBest(O)
  Simulation(Q)
  P←Select(P∪Q)
end while
```

The implementation of NSGA-II-ANN is described with pseudo code in Algorithm 4. No modifications of this algorithm have been done.

Algorithm 4 NSGA-II integrated with an artificial neural network (NSGA-II-ANN).

```
P←Create()
Simulation(P)
numSimulations←0
simulate←true
while (not StopOptimisation()) do
  O←∅
  repeat
    o←Crossover(TournamentSelection(P), TournamentSelection(P))
    Mutate(o)
    O←Add(o)
  until (λ offspring created)
  if simulate then
    Simulation(O)
  else
    SurrogateEvaluation(O)
  end if
  numSimulations←numSimulations+1
  if numSimulations > n then
    simulate←false
  else if numSimulations > m then
    simulate←true
    numSimulations←0
  end if
  P←Select(P∪Q)
end while
```

5.5 Surrogate configuration

As previously discussed in Section 2.3.1, there are two categories of surrogates: surrogate approximations and surrogate models. To repeat, a surrogate approximation is trained to learn the functional relationship between the output y and the input x of the simulation based on data samples obtained from previous runs of the simulation, while a surrogate model is a simplified version of the simulation. Surrogate approximations are by far the most commonly used of the two. Furthermore, surrogate approximations are generally easier to construct since neither knowledge of

the simulation internals nor programming skills is required. Instead, the user only specifies parameter values for the surrogate approximation and it learns to imitate the simulation by itself. Due to these advantages, surrogate approximations have been the primary choice in this work.

Comparative studies of the performance of different surrogate approximations have shown that there is no universally superior technique, but that the best choice depends on the characteristics of the problem under consideration. For real-world problems, the nature of the true function is not known a priori and it is therefore not possible to provide explicit rules on the selection of the most accurate surrogate technique (Jin et al., 2001; Queipo et al., 2005). Generally, however, multi-layered artificial neural networks have been considered the technique of preference in settings with complex function relationships and a limited number of data samples, as is common in real-world problems (Ratle, 1999; Jin, 2005). Several properties of artificial neural networks make them beneficial for use as surrogates, including universal approximation characteristics, good extrapolation/generalisation ability, applicability to multivariate non-linear problems, ability to handle noise in data sets, and no inherent assumption about data correlations. Artificial neural networks are also successfully being used in commercial optimisation software packages (Laguna and Marti, 2003). Due to their advantages, artificial neural networks have been used for surrogate approximation in the evaluation. It should, however, be noted that MOPSA-EA allows for any kind of surrogate and that another approximation technique could be used as well.

The artificial neural network used has one hidden layer, since it has been shown that one hidden layer is sufficient for nearly all problems (e.g. Chen et al. 1995). The number of hidden nodes in the artificial neural network is dynamically adapted depending on the number of samples available. For good performance, it is recommended that the number of weights in the artificial neural network is proportional to the size of the training data set (Mehrotra et al., 1996). Since the number of samples

continuously increases during the optimisation, a static number of hidden nodes is not appropriate. Therefore, the optimisation started with an artificial neural network with one single hidden node. When the number of available samples exceeded five times the number of weights in the network, a new hidden node was added (according to the weight-sample ratio suggested in Mehrotra et al. 1996). The artificial neural network was trained using back-propagation with the sigmoid function (Equation 2.5) and a learning rate of 0.5. For each 10th simulation, the artificial neural network was re-trained with samples from the most recently simulated solutions (at most 50 samples). In case any of these solutions had been simulated more than once, the mean simulation values were used in the training. The idea of regularly re-training the artificial neural network with the most recent samples is to have a local surrogate defined over the current search region. Local artificial neural networks are preferable to global artificial neural networks in surrogate-assisted optimisation (Giotis et al., 2000; Jin, 2005), since they reduce the computational time of the training process (Jin and Branke, 2005). To avoid overfitting, 10-folded cross validation was used in the training.

In the optimisation problem of Volvo Cars Engine (see Section 5.1.2), constructing a useful artificial neural network was not possible since the number of simulation inputs was very large (about 500) and dynamic. An artificial neural network with over 500 inputs includes tens of thousands of network weights, and such a network cannot perform well when the problem is complex and the number of data samples is limited. Consequently, for Volvo Cars Engine a surrogate model had to be used instead of an artificial neural network. The surrogate model was built using the C# programming language and solves the same problem as the simulation through a number of simplifications (for example, carts transporting camshaft between machines were not modelled). Since it is less complex than the simulation, it is also computationally cheaper and thereby serves the same purpose as an artificial neural network. Validations of the surrogate model show that it generally mimics the

simulation well, but that there are some cases where the output from the surrogate model and the simulation differs substantially. Analysis of these deviations showed that they were due to deadlocks in the machines in the simulation under certain conditions. Since the surrogate model and the real simulation model do not simulate the machines in exactly the same way, these deadlocks did not occur in the surrogate model.

5.6 Algorithm parameter settings

The performance of an evolutionary algorithm is dependent on its parameter settings (Ochoa et al., 1999), and therefore the parameter values of MOPSA-EA were tuned in the evaluation. Over the years, there have been a variety of research studies that investigated optimal parameter settings in evolutionary algorithms, and this issue is still an active area of research (Lobo et al., 2007). Although parameter configuration has been the subject of numerous studies, the best settings must usually be found through experimental examination of alternative values since the optimal configuration is dependent on the nature of the problem (Goldberg, 1989; Ochoa et al., 1999; Tran, 2006). Section 5.6.1 presents parameter settings for the ZDT benchmark problems, while the settings for the two real-world problems are given in Section 5.6.2.

5.6.1 Benchmark problems

In order to find a good parameter configuration for MOPSA-EA in the benchmark problems, three alternative settings were tested for each of its parameters:

- Population size: 20, 60, and 100.
- Number of offspring: 20, 60, and 100.
- Mutation step size[5]: 0.5, 1.0, and 1.5.
- Crossover: Single-point (SP), blended, and uniform.
- Crossover probability: 0.4, 0.6, and 0.8.

All combinations of the different settings were tested for all parameters, which means that a total of 243 experiments were performed for each of the five benchmark functions. Each experiment was replicated 100 times and the average values of the Υ, Ω, and S metrics were taken as the result. In finding the optimal parameter settings for a function, metric values from all the experiments were collected and ranked according to achieved value. By taking the sum of the achieved ranks of each metric, the best setting has been identified. The optimal settings found for the five functions are presented in Table 5.3.

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
Population size	20	20	60	20	20
Number of offspring	60	100	60	100	60
Mutation step size	0.5	0.5	0.5	0.5	1.0
Crossover	SP	SP	SP	SP	SP
Crossover probability	0.8	0.6	0.8	0.8	0.8

Table 5.3: Optimal parameter settings for MOPSA-EA in benchmark problems.

In order to conduct a fair performance comparison, tuning of parameter settings was also applied to m-SMS-EMOA, m-MAES, and NSGA-II-ANN. Tables 5.4-5.6 show the optimal parameter settings found for these algorithms for the five benchmark functions. In m-MAES, the number of surrogate evaluated offspring candidates in a generation is *population size*$\cdot$ *number of offspring* (i.e. *number of offspring* denotes the

[5] The mutation step size defines the deviation of the multivariate normal distribution used in the mutation operator. The number of dimensions of the distribution equals the number of input parameters to the problem. Note that the probability of a mutation occurring is 0.1.

number of candidates per ordinary offspring) in order to allow the same total number of offspring candidates in the different algorithms. Note that NSGA-II-ANN does not make use of offspring candidates and therefore *number of offspring* does not apply to this algorithm. The surrogate configuration of m-SMS-EMOA, m-MAES, and NSGA-II-ANN has been the exact same as for MOPSA-EA (c.f Section 5.5).

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
Population size	20	20	20	20	60
Number of offspring	60	100	60	100	60
Mutation step size	1.5	1.5	1.5	1.5	1.5
Crossover	SP	Blend	SP	SP	SP
Crossover probability	0.8	0.6	0.8	0.8	0.6

Table 5.4: Optimal parameter settings for m-SMS-EMOA in benchmark problems.

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
Population size	100	100	100	100	100
Number of offspring	100	100	100	100	60
Mutation step size	0.5	0.5	0.5	0.5	1.0
Crossover	SP	Blend	SP	SP	SP
Crossover probability	0.4	0.4	0.8	0.8	0.6

Table 5.5: Optimal parameter settings for m-MAES in benchmark problems.

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
Population size	60	60	60	100	100
Mutation step size	0.5	0.5	0.5	0.5	0.5
Crossover	Blend	Blend	SP	Blend	Blend
Crossover probability	0.6	0.4	0.8	0.6	0.4

Table 5.6: Optimal parameter settings for NSGA-II-ANN in benchmark problems.

5.6.2 Real-world problems

In the real-world problems, thorough parameter pre-tuning could not be carried out within a practical time frame due to the computational time of the simulations used to evaluate solutions. Instead, for these problems the parameter values were set in discussion with system experts on the basis of domain knowledge. The system experts (simulation technicians and production engineers) were introduced to the

various parameters and asked to provide proper values based on their knowledge and a limited number of simulation tests. Discussions about the parameter settings then took place in physical meetings and over the phone. The resulting configuration of the parameters is presented in Table 5.7. Note that MOPSA-EA, m-SMS-EMOA, m-MAES, and NSGA-II-ANN all use the same parameter settings.

	Volvo Aero	Volvo Cars Engine
Population size	20	30
Number of offspring	20	30
Probability of mutation	0.1	0.1
Mutation step size	1.0	See Section 5.1.2
Crossover operator	Single-point	Uniform
Crossover probability	0.8	0.8

Table 5.7: Algorithm parameter settings in real-world problems.

5.7 Noise handling

In the implementation of the confidence-based dynamic resampling (CDR) technique used in MOPSA-EA, two parameters must be specified: confidence level and maximum number of samplings. Section 5.7.1 describes how these parameters were set in the evaluation. To assess the relative performance of the CDR technique, it was compared to existing techniques to deal with noise, as presented in Section 5.7.2. These techniques were also integrated and tested with MOPSA-EA to compare the results.

In Section 3.5.2, two potential improvements in the implementation of the CDR technique were discussed. The first one was to check if the two solutions being compared are significantly different with respect to the highest confidence level of rank, and thereby avoid a non-dominating sort to be performed unnecessarily. The second one was to perform less resamplings in the beginning of the search, and increase the resampling frequency when the optimisation begins to converge. In the evaluation, CDR was implemented with the extra confidence check, but not with the variable resampling frequency in order to avoid additional user-defined parameters.

5.7.1 Parameter settings

Tuning the parameters of the CDR technique to find their best values would require a huge number of experiments. Not only must all combinations of the CDR's parameters be tested, but these must also be tested in combination with all parameters of the algorithm (e.g. population size and mutation rate). Each of these tests must also be replicated due to noise. Since such extensive testing is usually not affordable, it is necessary that the CDR technique is not too sensitive with respect to its parameter settings. Therefore, no parameter tuning was carried out for the evaluation, but the technique was tested with the same parameter set-up in all problems. These settings are listed in Table 5.8 and Table 5.9. As can be seen, a maximum of five samples per solution was used. This is a relatively low number of samplings. However, considering that real-world scenarios only allow a limited number of simulations within the optimisation time-budget (often only a few hundreds), only a small number of evaluations of each solutions could be carried out.

	Confidence level
Rank 1	0.75
Rank 2	0.70
Rank 3	0.65
Rank 4	0.60
Rank 5 and higher	0.55

Table 5.8: Confidence levels for different ranks.

	Max samplings
Rank 1	5
Rank 2	4
Rank 3	3
Rank 4 and higher	2

Table 5.9: Maximum number of samplings for different ranks.

5.7.2 Performance comparison

As described in Section 4.3, there are four major techniques for handling noise: (*i*) static resampling, (*ii*) modified Pareto ranking scheme, (*iii*) dominance-dependent

lifetime, and (*iv*) fitness inheritance. The CDR technique was compared to all four of these. The implementation of the techniques in the evaluation is described below. It is important to note out that with all techniques (including CDR), the total number of simulations performed in an optimisation run is the same. Consequently, the number of unique solutions evaluated may vary with the different techniques; given n simulations and s samplings of each solution, n/s unique solutions are evaluated (since one simulation equals one sampling of a solution).

Static resampling

In the implementation of this technique, each solution is sampled five times and the average objective values are used. This is the same number of samplings used at maximum in the CDR technique, which means that the noise reduction of the static resampling scheme will never be worse than that of CDR.

Modified Pareto ranking scheme

There are two different approaches for modifying the original Pareto ranking scheme to handle noise; the probability-based technique and the cluster-based technique. The main idea of the former is to use probabilistic ranks based on dominance relations instead of the ordinary Pareto ranks based on objective values. With probabilistic ranks, most solutions will be assigned unique ranks as a consequence of the ranks being real values. A potential problem with this is that crowding distance among solutions (i.e. the density of solutions in a particular area) is not be considered, resulting in a poor diversity of the population. Another drawback of the probability-based technique is the computational expense from performing integral operations for each objective in each solution (cf. Equation 4.5). Considering these drawbacks, the second technique of modifying the Pareto ranking scheme, that is cluster-based ranking, was used in the evaluation performed in this study.

The cluster-based technique involves the user-defined parameter K, defining the neighbourhood restriction factor. Babbar et al. (2003) set K according to Equation 5.2

$$K = C * \left(1 - e^{-\frac{\beta}{Ge}}\right), \tag{5.2}$$

where C and β are problem-specific constants, and Ge is the current generation. In this study, the technique is implemented in a steady-state algorithm and Ge is therefore undefined. K can therefore not be set according to Equation 5.2, but the formula is therefore adjusted to fit a steady-state approach according to Equation 5.3

$$K = C * (maxSim - numSim), \tag{5.3}$$

where C is a constant set to 0.0001 (a value that has to be found through experimental tuning since no information about appropriate C values was available), $maxSim$ is the maximum number of simulations allowed for the optimisation (a constant value), and $numSim$ is the number of simulations performed at the current point in time (a variable number). Similar to Equation 5.2, the neighbourhood decreases over time with Equation 5.3.

Dominance-dependent lifetime

In the implementation of this technique, the lifetime lt of a new solution i entering the population is set according to Equation 5.4

$$i_{lt} = popSize - i_{nd}, \tag{5.4}$$

where $popSize$ is the population size, and nd is the number of solutions i is dominating. Equation 5.4 ensures that a short lifetime is assigned if a large number of solutions in the population are dominated and vice versa. A non-dominating solution whose lifetime has expired is resampled once and added to the population with its new objective values.

Fitness inheritance

With this technique, a child inherits the mean objective value $\bar{\mu}$ and mean standard deviation $\bar{\sigma}$ of each objective from its parents. The child is evaluated once, and if each obtained objective value falls within the confidence interval $\left(\bar{\mu} - 3 * \bar{\sigma} \leq f \leq \bar{\mu} + 3 * \bar{\sigma}\right)$, the inherited fitness is accepted. Otherwise, the child is resampled four times and assigned the mean values of its samplings.

5.8 Parallel performance

Since MOPSA-EA is designed for parallel execution it is necessary to evaluate its parallel performance. Parallel qualities of evolutionary algorithms are difficult to analyse theoretically and are therefore generally evaluated empirically (Alba and Luque, 2006). Among the most commonly used metrics in empirical evaluations is speedup, since the primary aim of parallelising an algorithm is to reduce its computational time (Alba and Luque, 2006). *Speedup* captures the relative benefit of performing an optimisation in parallel and is the time it takes to complete the optimisation with the fastest processing node divided by the time it takes with n nodes (Karp and Flatt, 1990). Equation 5.5 defines the speedup S_n achieved with n processing nodes:

$$S_n = \frac{t(f)}{t(n)}, \tag{5.5}$$

where $t(f)$ is the time the optimisation takes when using the fastest node f , and $t(n)$ is the time it takes when using all n nodes. Linear (ideal) speedup is obtained when $S_n = n$. For measuring speedup, wall-clock time is used rather than CPU time. This is one to include the communication overhead introduced by running an algorithm in parallel (Alba and Luque, 2006).

In addition to speedup, efficiency is an important metric for the evaluation of a parallel algorithm (Veldhuizen et al., 2002). *Efficiency* is a measurement of how well an

algorithm is utilising the processing nodes and is calculated by dividing its speedup by the number of processing nodes. Equation 5.6 defines the efficiency E_n achieved with n processing nodes:

$$E_n = \frac{S_n}{n}, \tag{5.6}$$

where $S_n$is the speedup as defined in Equation 5.5. An efficiency of 1 means that all of the processing nodes are being fully used all the time.

Speedup and efficiency of MOPSA-EA were measured on the Volvo Aero problem (see Section 5.1.2), which includes 400 simulation evaluations. To prevent stochastic events of the simulation from influencing the measurements, all such events were disabled (i.e. the simulation was run deterministically). Five different setups were tested with different numbers of processing nodes: 1, 10, 20, 30, and 40 nodes. Note that this is the number of slave nodes; there will also be an additional master node (see the description of the master-slave model in Section 2.2.1). The experiments were carried out using the OPTIMISE platform (see Section 5.2) and a cluster of homogeneous processing nodes.

5.9 Summary

This chapter presented the evaluation of MOPSA-EA on five ZDT benchmark problems and two real-world problems of manufacturing optimisation. The first real-world problem concerns a manufacturing cell for the production of aircraft and gas turbine engine components at Volvo Aero, while the second problem concerns a camshaft machining line at Volvo Cars Engine. In the Volvo Aero problem, an artificial neural network was adopted as a surrogate evaluation function, while in the Volvo Cars Engine problem a surrogate model built using the C# programming language was used. To assess the relative performance of MOPSA-EA, the algorithm was compared to three existing, surrogate-assisted, multi-objective evolutionary algorithms: SMS-

EMOA, NSGA-II, and MAES. The novel noise handling technique used in MOPSA-EA was also compared with four other techniques for dealing with noise: static resampling, modified pareto ranking scheme, dominance dependent lifetime, and fitness inheritance. To evaluate the parallel performance of MOPSA-EA, its speedup and efficiency with increasing numbers of processors were measured. All experiments in the evaluation were realised using the OPTIMISE platform, which was also presented in the chapter.

In the next chapter, the results from the evaluation are presented and an analysis of the optimisations is provided.

Chapter 6

Results and analysis

This chapter presents and discusses the results from the evaluation of MOPSA-EA. The algorithm was evaluated both with and without consideration to noise. Although noise handling is an important feature of the algorithm, it is easier to test and analyse its fundamental design principles without noise. The optimisation results without considering noise are presented in Section 6.1, while Section 6.2 presents the results when considering noise. In Section 6.3, the results from the evaluation of the new noise handling technique used in MOPSA-EA are presented. A confidence probability has been calculated for all optimisations using Welch's t-test (Law and Kelton, 2000) to indicate the probability of the true result matching the presented results. The minimum confidence level for the benchmark problems was set to 99%, while the minimum confidence level for the real-world problems was set to 80%[1]. At the conclusion of the chapter (Section 6.4), a description of the parallel performance of the algorithm is presented.

[1] A lower confidence level is set in the real-world optimisations due to practical reasons; reaching a high confidence level requires a very large number of optimisation runs, and that many runs was not possible to complete within a reasonable amount of time.

6.1 Optimisation without consideration to noise

This section presents the results of MOPSA-EA when applied without consideration to noise. In addition, comparative results of m-SMS-EMOA, m-MAES, and NSGA-II-ANN are also presented. Section 6.1.1 describes the benchmark optimisations, while Sections 6.1.2 and 6.1.3 describe the optimisation of the two real-world problems. An overall analysis of the results is given in Section 6.1.4.

6.1.1 Results benchmark functions

Results from the benchmark functions are shown in Table 6.1. The optimisation of each function was performed for 5000 function evaluations. Furthermore, the results in Table 6.1 are based on normalised objective values and constitute the average of 500 independent runs. In calculating the Υ and IGD metrics, a set of 500 uniformly-distributed solutions of the true Pareto front was derived. The S metric requires a reference point, which was set to (x,y)-coordinates that are just outside the maximum values in objective space taken by the respective functions, according to Table 6.2. Based on the reference point, the S value was normalised between [0,1].

As shown in Table 6.1, MOPSA-EA achieves the best results on all ZDT functions and NSGA-II-ANN the second best. m-MAES generally achieves the third best results, while m-SMS-EMOA generally achieves the worst.

	ϒ (minimise)	IGD (minimise)	S (maximise)
ZDT1			
MOPSA-EA	0.129	0.091	0.896
m-SMS-EMOA	0.193	0.154	0.706
m-MAES	0.175	0.118	0.804
NSGA-II-ANN	0.158	0.104	0.827
ZDT2			
MOPSA-EA	0.334	0.312	0.786
m-SMS-EMOA	0.64	0.585	0.495
m-MAES	0.573	0.622	0.619
NSGA-II-ANN	0.424	0.401	0.691
ZDT3			
MOPSA-EA	0.164	0.065	0.932
m-SMS-EMOA	0.516	0.279	0.742
m-MAES	0.338	0.21	0.817
NSGA-II-ANN	0.225	0.149	0.894
ZDT4			
MOPSA-EA	0.836	0.3	0.884
m-SMS-EMOA	1.402	1.086	0.811
m-MAES	1.612	0.988	0.84
NSGA-II-ANN	1.206	0.502	0.859
ZDT6			
MOPSA-EA	0.576	0.537	0.718
m-SMS-EMOA	0.815	0.631	0.713
m-MAES	0.751	0.704	0.714
NSGA-II-ANN	0.674	0.571	0.716

Table 6.1: Benchmark results.

Function	x	y
ZDT1	1.5	8.0
ZDT2	1.5	8.0
ZDT3	1.5	8.0
ZDT4	1.5	250.0
ZDT6	1.5	150.0

Table 6.2: Reference points for ZDT functions.

6.1.2 Results Volvo Aero

Table 6.3 presents the results from the optimisation of the Volvo Aero problem. The result values constitute the average of ten independent runs (due to the computational expense associated with the simulation, only a relatively small number of runs could

be undertaken). Since the true Pareto-optimal front of the Volvo Aero problem is unknown, as with real-world problems in general, only the S metric could be calculated. The optimisation was performed for 400 simulations and all algorithms start from the same initial population. As shown in the table, MOPSA-EA achieves the best value, NSGA-II-ANN the second best, m-MAES the third best and m-SMS-EMOA the worst. This ranking order is the same as in the benchmark functions.

	S (maximise)
MOPSA-EA	0.465
m-SMS-EMOA	0.374
m-MAES	0.426
NSGA-II-ANN	0.446

Table 6.3: Optimisation results in Volvo Aero problem.

To visualise the results of the algorithms, median summary attainment surfaces are shown in Figure 6.1. A summary attainment surface is a visual way of summarising a number of runs of a multi-objective algorithm, based on the notion of an attainment surface (Knowles, 2005). Given a set of non-dominated solutions produced by a single run of an algorithm, the attainment surface is the boundary in objective space that separates the region dominated by, or equal to, this set from the region that is non-dominated. The interpretation of the median attainment surface is that, for every point on it, a point dominating this was obtained in at least 50% of the non-dominated sets.

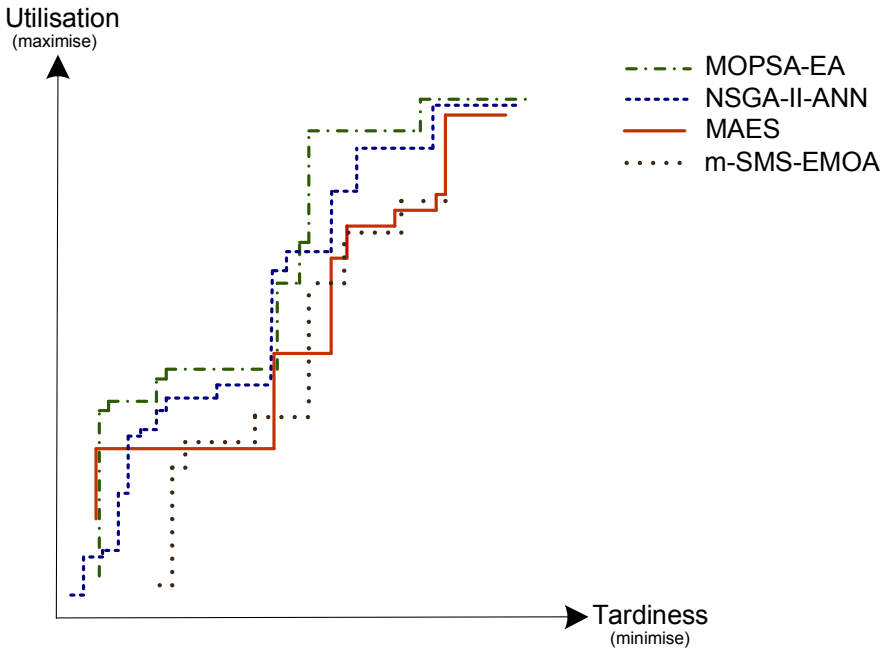


Figure 6.1: Median attainment surface achieved by each algorithm in Volvo Aero optimisation.

In Appendix G, the optimisation results are further elaborated on and the training of the artificial neural network surrogate used in the optimisation is discussed.

6.1.3 Results Volvo Cars Engine

The results of the optimisation of the Volvo Cars Engine problem are presented in Table 6.4 and Figure 6.2 (average of ten independent runs). The optimisation was performed for 600 simulations. As in the Volvo Aero problem, the true Pareto-optimal front is unknown and only the S metric was calculated (note that although they look similar, the S values in Table 6.4 cannot be compared to those of the Volvo Aero problem due to different settings of the reference points). The results show that the ranking order of the algorithms is the same as in the Volvo Aero problem. However, the gap between MOPSA-EA and NSGA-II-ANN is less in this problem than in the Volvo Aero problem. This might be explained by the different surrogate techniques used in the two problems. In NSGA-II-ANN the approach of alternating between the simulation and the surrogate used in evaluating generations may be subject to strong oscillation

if the surrogate if of poor quality (Jin, 2005). The artificial neural network surrogate used in the Volvo Aero problem initially has large errors that slowly decrease when the training set becomes bigger, while the surrogate model used in this problem is static and therefore has the best possible quality right from the beginning, possibly giving NSGA-II-ANN a better search start.

	S (maximise)
MOPSA-EA	0.481
m-SMS-EMOA	0.408
m-MAES	0.441
NSGA-II-ANN	0.471

Table 6.4: Optimisation results in Volvo Cars Engine problem.

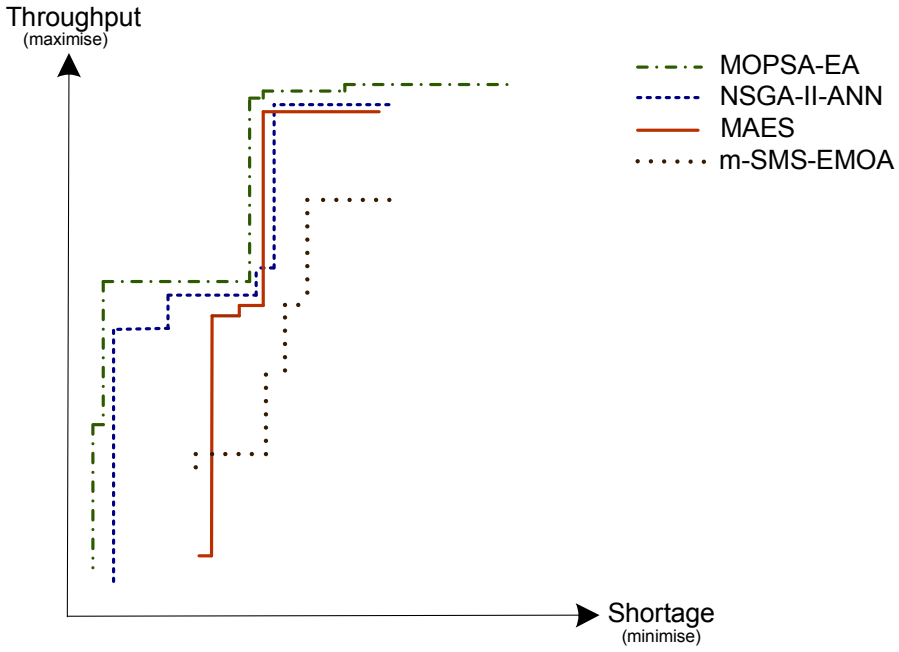


Figure 6.2: Median attainment surface achieved by each algorithm in Volvo Cars Engine optimisation.

The optimisation results for the Volvo Cars Engine problem are further elaborated in Appendix H.

6.1.4 Analysis

There are three characteristics of MOPSA-EA that together make this algorithm unique among the algorithms used in the comparative evaluation: the steady-state design, the selection strategy, and the compensation for surrogate imprecision. In this section, these characteristics are further discussed along with their possible contribution to the successful results of MOPSA-EA.

Steady-state design

In the literature, steady-state evolutionary algorithms are reported to perform better than generational evolutionary algorithms when the search space is complex and the number of solution evaluations is limited due to time-consuming evaluations (e.g. Chafekar et al. 2005; Lacksonen 2001; Jones and Soule 2006; Lozano et al. 2008). Three main aspects of the steady-state approach have been identified as contributing to a more aggressive shift towards the optimum in these algorithms (Vavak and Fogarty, 1996; Chafekar et al., 2005):

(i) High preservation of well-performing solutions (only the worst solution is replaced in every iteration and good solutions therefore survive for many iterations)

(ii) High selection pressure (two selections are performed in every iteration; insertion and deletion)

(iii) New solutions are immediately used as a part of the mating pool.

However, if MOPSA-EA is compared to m-MAES and NSGA-II-ANN (both generational) with respect to these aspects, the differences are smaller than they may seem. NSGA-II, the base of both m-MAES and NSGA-II-ANN, differs from an ordinary generational algorithm in that it uses an elitistic approach in which the next generation of the population is formed by the best solutions in the combined set of parents and

offspring. This approach promotes a high preservation of well-performing solutions. The strong elitism in NSGA-II has also shown to cause a high selection pressure (Deb and Goel, 2001). These aspects are thus not unique to MOPSA-EA. However, allowing new individuals to be immediately available for mating is unique to MOPSA-EA in comparison to m-MAES and NSGA-II-ANN, and might have had some influence on its better performance. It is hard to estimate *how* much impact this latter aspect actually has, and no previous studies analysing this aspect of steady-state algorithms in isolation could be found. The only conclusion that can be drawn is that the steady-state design is not the only factor contributing to the good results of MOPSA-EA, considering that m-SMS-EMOA is also based on a steady-state design and achieves the worst results in this study. In summary, the steady-state design may not have any significant impact on the convergence of MOPSA-EA, but it is still an important feature of the algorithm with respect to parallel efficiency.

Selection strategy

A factor playing a more decisive role in the better performance of MOPSA-EA may instead be the offspring and replacement selection strategy. The selection strategy adopted in an EA has a significant impact on the diversity of the population and, thereby, also on the convergence of the algorithm since poor diversity may lead to local optima. The selection strategy used in MOPSA-EA differs from that used in m-MAES and m-SMS-EMOA mainly with respect to the ranking of solutions located on the same Pareto front. While MOPSA-EA uses the crowding distance measure in this ranking, which is meant to distribute solutions uniformly, m-MAES and m-SMS-EMOA use the hypervolume measure, which is meant to distribute them in a way that maximises the hypervolume. The idea of the hypervolume strategy is to promote solutions with well-balanced objective values, that is, to avoid solutions with a little gain in one objective at the price of a large sacrifice in the other objective (Emmerich et al., 2005). The difference in outcome of the two ranking strategies is exemplified in

Figure 6.3. In this example, a selection between solution A and B would result in A if the crowding distance measure is considered, and B if the hypervolume measure is considered instead (assuming both f_1 and f_2 are to be minimised).

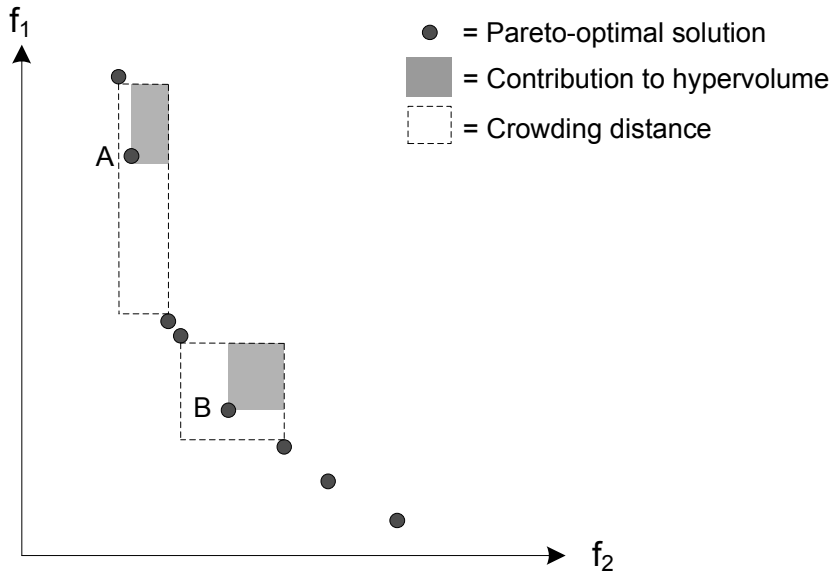


Figure 6.3: Crowding distance vs. hypervolume.

The hypervolume-based selection strategy has previously been analysed in Emmerich et al. (2005) on the ZDT functions. This study indicates that the strategy concentrates on a few solutions near the "knee-points" of the Pareto front, and that flanks of the front with unbalanced trade-offs between objectives are covered with less density. The hypervolume-based selection strategy being biased towards a few solutions located in specific areas of the search space might lead to poor diversity of the population. In comparison, the crowding distance-based strategy used in MOPSA-EA explicitly aims at selecting as diverse solutions as possible, and since this strategy is applied twice in every step of the algorithm (offspring and replacement selection) a high diversity in the population is strongly promoted. The higher population diversity in MOPSA-EA compared to m-MAES and m-SMS-EMOA can be an explanation of the superior results of the former. This theory is supported by the fact that NSGA-II-ANN, which also uses crowding distance in selections, achieves good results in the evaluations.

Compensating for surrogate imprecision

A further aspect potentially explaining the better performance of MOPSA-EA is that the imprecision of the surrogate is taken into consideration in the offspring selection procedure of the algorithm. As previously discussed in Section 2.3.1, surrogates seldom represent the search space correctly and if this is not compensated for the evolutionary algorithm may converge towards false optima. To get an idea of the impact of surrogate imprecision, MOPSA-EA was also applied to the ZDT function and the two real-world problems without such compensation (i.e. offspring are simply selected based on the given surrogate objective values). The results of this optimisation in comparison to the original results (i.e. when considering surrogate imprecision) are shown in Tables 6.5 and 6.6. The results clearly indicate that MOPSA-EA obtains better results when surrogate imprecision is considered, especially for the real-world problems for which it is harder to construct a surrogate of high accuracy.

	Υ (a) (minimise)	Υ (b) (maximise)	IGD (a) (minimise)	IGD (b) (minimise)	S(a) (maximise)	S(b) (maximise)
ZDT1	0.129	0.158	0.091	0.102	0.896	0.867
ZDT2	0.334	0.39	0.312	0.378	0.786	0.766
ZDT3	0.164	0.169	0.065	0.074	0.932	0.928
ZDT4	0.836	1.06	0.3	0.402	0.884	0.876
ZDT6	0.576	0.711	0.537	0.693	0.718	0.717

Table 6.5: Results of MOPSA-EA with (a) and without (b) compensation for surrogate imprecision in ZDT problems.

	S(a) (maximise)	S(b) (maximise)
Volvo Aero	0.465	0.44
Volvo Cars Engine	0.481	0.467

Table 6.6: Results of MOPSA-EA with (a) and without (b) compensation for surrogate imprecision in real-world problems.

6.2 Optimisation with consideration to noise

The performance of MOPSA-EA was compared to that of m-SMS-EMOA, m-MAES and NSGA-II-ANN when noise is considered in the optimisation. For a fair comparison, the noise must not only be reduced in MOPSA-EA, but also in the other three algorithms. Therefore, m-SMS-EMOA, m-MAES, and NSGA-II-ANN adopt the static resampling scheme in which all solutions are sampled five times (see Section 5.7.2). This is the same number of samplings that MOPSA-EA spends on a solution at maximum, which means that the noise reduction of m-SMS-EMOA, m-MAES and NSGA-II-ANN will never be more than that of MOPSA-EA. In the optimisations, MOPSA-EA was also tested without using the confidence-based dynamic resampling (CDR) technique, but with the static resampling scheme instead. Note that in all five algorithms the total number of simulations used in an optimisation was the same (given n simulation replications and s samplings, n/s unique solutions are evaluated). In addition, all algorithms start from the same initial population.

Regarding the ZDT functions, these are deterministic and not primarily constructed to be used to evaluate aspects related to noise. However, for the sake of replicability and verifiability, one of these functions was used in the evaluation with artificial noise added. Without any bias, the first function (ZDT1) was simply selected. Artificial noise for this function was generated from a zero mean Gaussian distribution whose standard deviation represents the amount of noise. Three different amounts of noise were used: 0.1, 0.15, and 0.2 (similar amounts of noise are used in several other studies to simulate noise in test functions, see for example Babbar et al. 2003; Bui et al. 2004; Goh and Tan 2006). Results from the optimisation of the noisy ZDT1 function are presented in Section 6.2.1. Sections 6.2.2 and 6.2.3 describe the results of the Volvo Aero and Volvo Cars Engine problems. A concise analysis of the results is given in Section 6.2.4.

6.2.1 Results ZDT1

Results from the noisy version of the ZDT1 function are shown in Table 6.7 (average of 500 independent runs). The optimisation was performed for 5000 function evaluations and the results are based on normalised objective values. As with the noise-free version of the function, a set of 500 uniformly distributed solutions of the true Pareto front was used for calculating the Υ and IGD metrics.

As shown in Table 6.7, MOPSA-EA achieves the best results with all three amounts of noise. MOPSA-EA without the CDR technique is the second best algorithm, while NSGA-II-ANN and m-MAES share third place. Similar to the noise-free optimisation, m-SMS-EMOA generally achieves the worst results.

	Υ (minimise)	IGD (minimise)	S (maximise)
Noise 0.1			
MOPSA-EA	0.158	0.1	0.84
MOPSA-EA (no CDR)	0.185	0.121	0.8
m-SMS-EMOA	0.27	0.188	0.657
m-MAES	0.21	0.137	0.696
NSGA-II-ANN	0.164	0.139	0.654
Noise 0.15			
MOPSA-EA	0.168	0.111	0.815
MOPSA-EA (no CDR)	0.2	0.134	0.752
m-SMS-EMOA	0.281	0.197	0.626
m-MAES	0.217	0.14	0.669
NSGA-II-ANN	0.22	0.146	0.664
Noise 0.2			
MOPSA-EA	0.207	0.138	0.802
MOPSA-EA (no CDR)	0.22	0.158	0.748
m-SMS-EMOA	0.3	0.21	0.664
m-MAES	0.272	0.226	0.681
NSGA-II-ANN	0.236	0.177	0.673

Table 6.7: Results ZDT1 with noise.

6.2.2 Results Volvo Aero

Results from optimising the Volvo Aero problem with noise reduction are presented in Table 6.8 and Figure 6.4 (average of ten independent runs). The optimisation was

performed for 400 simulations. In calculating the S value, the Pareto front found by the algorithm was replicated 20 times and the mean values of the simulation replications were used. As shown from the results, MOPSA-EA achieves the best results, MOPSA-EA without the CDR technique the second best, NSGA-II-ANN the third best, m-MAES the fourth best, and m-SMS-EMOA the worst. Compared to the results without noise reduction (see Section 6.1.2), it is noticeable that all algorithms benefit from performing multiple samplings of solutions. The results of m-MAES improved considerably; the S value obtained by this algorithm increases from 0.426 to 0.451.

	S (maximise)
MOPSA-EA	0.496
MOPSA-EA (no CDR)	0.467
m-SMS-EMOA	0.398
m-MAES	0.451
NSGA-II-ANN	0.452

Table 6.8: Results Volvo Aero problem with noise.

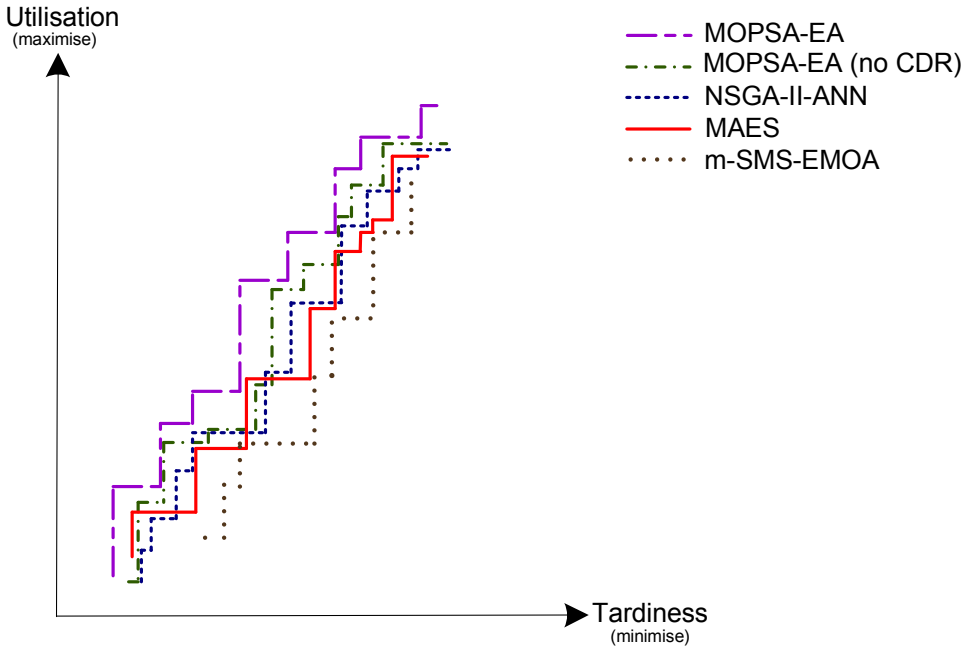


Figure 6.4: Median attainment surface achieved by each algorithm in Volvo Aero optimisation (with noise compensation).

6.2.3 Results Volvo Cars Engine

The optimisation results from the Volvo Cars Engine problem with noise reduction are presented in Table 6.9 and Figure 6.5 (average of ten independent runs). The optimisation was performed for 600 simulations. The S metric was calculated in the same way as in the previously described Volvo Aero problem. As shown in Table 6.9, the algorithms rank in the same order as in the Volvo Aero problem. Also in the Volvo Cars Engine problem, it is noticeable that all algorithms benefit from performing multiple sampling of solutions (see Section 6.1.3).

	S (maximise)
MOPSA-EA	0.511
MOPSA-EA (no CDR)	0.483
m-SMS-EMOA	0.44
m-MAES	0.469
NSGA-II-ANN	0.479

Table 6.9: Results Volvo Cars Engine optimisation with noise compensation.

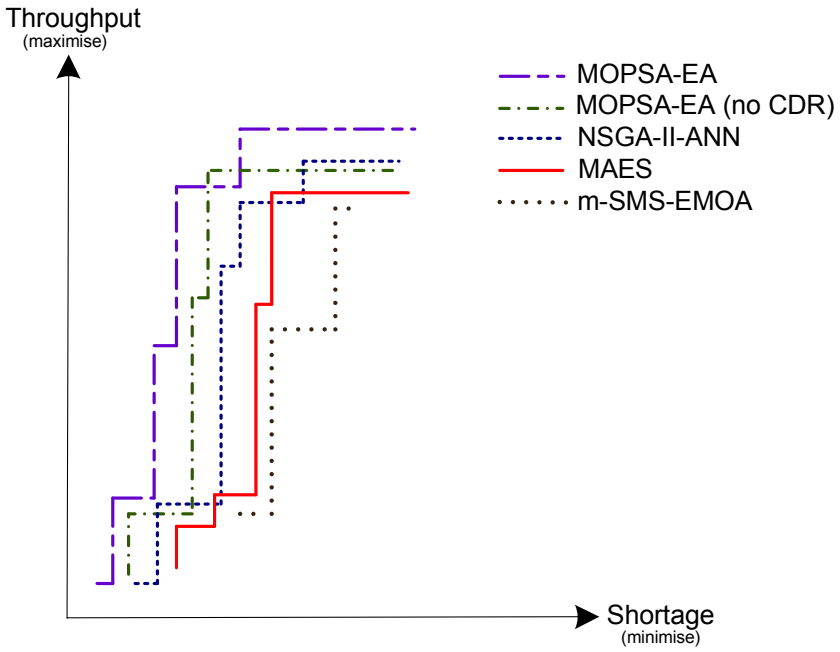


Figure 6.5: Median attainment surface achieved by each algorithm in Volvo Cars Engine optimisation (with noise compensation).

6.2.4 Analysis

The optimisation results show that the performance of MOPSA-EA is superior to the other algorithms, especially in the real-world problems that are subject to significant complex noise. This indicates that the CDR technique efficiently reduces noise. It must, however, be pointed out that when the noise is only nominal, it is generally better to use MOPSA-EA without this technique. This is because a small amount of noise has no significant influence on the optimisation, and simulation evaluations are then better spent on new solutions than on resampling old ones. This can be seen when both versions of the algorithm are applied on the ZDT1 function with amount of noise 0.001. As shown in Table 6.10, with nominal noise, MOPSA-EA without noise compensation achieves superior results. In the next section, noise compensation and the CDR technique are further elaborated.

	Υ (minimise)	IGD (minimise)	S (maximise)
MOPSA-EA (CDR)	0.197	0.161	0.701
MOPSA-EA (no CDR)	0.13	0.093	0.896

Table 6.10: Results of MOPSA-EA with (a) and without (b) the CDR technique on the ZDT1 function with amount of noise 0.001.

6.3 Noise handling

This section presents the results of evaluating the novel CDR technique in comparison to the techniques of static resampling, dominance-dependent lifetime, cluster-based Pareto ranking, and fitness inheritance. All noise handling techniques were integrated and tested with MOPSA-EA for comparative results. Section 6.3.1 describes the optimisation results for the ZDT1 function, while Sections 6.3.2 and 6.3.3 describe the results for the two real-world problems. In all three problems the performance metrics of the optimisations were calculated in the same way as in the previous evaluations. An analysis of the results is presented in Section 6.3.4.

6.3.1 Results ZDT1

Results from the ZDT1 function are shown in Table 6.11 (average of 500 independent runs). The results are based on normalised objective values. The optimisation was performed for 5000 function evaluations. As Table 6.11 illustrates, the best results are achieved with the CDR technique. The techniques of static resampling and cluster-based Pareto ranking achieve about the same results and share second place. Dominance-dependent lifetime is generally the fourth best technique, whilst the fitness inheritance technique achieves the worst results.

	Υ (minimise)	IGD (minimise)	S (maximise)
Noise 0.1			
CDR	0.158	0.1	0.84
Static resampling	0.185	0.134	0.8
Dominance-dependent lifetime	0.193	0.173	0.666
Cluster-based Pareto ranking	0.183	0.147	0.718
Fitness inheritance	0.283	0.151	0.676
Noise 0.15			
CDR	0.168	0.111	0.815
Static resampling	0.2	0.121	0.752
Dominance-dependent lifetime	0.202	0.168	0.622
Cluster-based Pareto ranking	0.196	0.166	0.696
Fitness inheritance	0.23	0.159	0.663
Noise 0.2			
CDR	0.207	0.138	0.802
Static resampling	0.22	0.158	0.748
Dominance-dependent lifetime	0.217	0.172	0.639
Cluster-based Pareto ranking	0.213	0.16	0.66
Fitness inheritance	0.331	0.196	0.637

Table 6.11: Results ZDT1

6.3.2 Results Volvo Aero

Results from the optimisation of the manufacturing cell at Volvo Aero is presented in Table 6.12 (average of ten independent runs). The optimisation was performed for 400 simulations. As Table 6.12 demonstrates, the best results are achieved with CDR,

second best with cluster-based Pareto ranking, third best with static resampling, fourth best with dominance-dependent lifetime, and the worst with fitness inheritance.

	S (maximise)
CDR	0.496
Static resampling	0.467
Dominance-dependent lifetime	0.411
Cluster-based Pareto ranking	0.468
Fitness inheritance	0.393

Table 6.12: Results Volvo Aero problem.

6.3.3 Results Volvo Cars Engine

Results from the optimisation of the camshaft machining line at Volvo Cars Engine are shown in Table 6.13. The optimisation was performed for 600 simulations and the results are an average from ten replications. As shown in Table 6.13, the noise handling techniques rank in the same order as in the Volvo Aero problem.

	S (maximise)
CDR	0.511
Static resampling	0.483
Dominance-dependent lifetime	0.409
Cluster-based Pareto ranking	0.485
Fitness inheritance	0.398

Table 6.13: Results Volvo Cars Engine problem.

6.3.4 Analysis

This section contains the discussion of analysis of the results of the five noise handling techniques. The section also includes an analysis of the sensitivity of the user-defined parameters of the CDR technique.

Performance comparison

In the optimisations, the CDR technique has the best performance in all three problems. A key factor contributing to its good results is most likely to be the dynamic resampling approach. The other techniques, except dominance dependent lifetime, use a static strategy in which all solutions entering the population are resampled a fixed number of times. Such a static approach is inefficient when considering that: (*i*) the same number of simulations are spent on solutions of higher ranks as on inferior solutions of lower ranks, (*ii*) all solutions are simulated the same number of times although they are subject to different amounts of noise[2], and (*iii*) simulations are wasted on solutions that never take part in the evolutionary process, that is, solutions which never influence the search. In contrast, the proposed technique uses a resampling scheme in which the number of samplings is automatically adjusted to the rank and noise of a solution. Furthermore, only solutions that are part of the evolutionary process are resampled. In this way, an efficient utilisation of simulations is achieved and the desired confidence level is reached using as few resamplings as possible.

The second best results in the optimisations are achieved by the techniques of static resampling and cluster-based Pareto ranking (these two achieve about the same overall results). The technique of static resampling is surprisingly efficient, especially in the real-world problems. Static resampling has previously been considered inappropriate when the number of simulations is limited due to their high computational cost (Branke et al., 2001; Branke and Schmidt, 2003). However, with respect to its simplicity and relative efficiency, it can be argued that this technique should not be rejected. In comparison, the technique of cluster-based Pareto ranking is more complex, but its performance is about the same. One explanation of the average results of the cluster-based technique might be that the diversity of the population is impeded with the

[2] Note that this applies only to the real-world problems – in the ZDT problems all solutions are subject to the same amount of noise.

modified ranking scheme. The clustering effect arising from including solutions close to solutions of rank 1 in the front results in many similar solutions. A consequence of many solutions being similar to each other is that the diversity of the population becomes poor and hence the convergence of the search is negatively affected.

Ranked as number four in most of the optimisations is the technique of dominance-dependent lifetime. Contrary to the other techniques, this one does not explicitly resample solutions, but instead evaluates non-dominating solutions whose lifetime has expired anew and adds them to the population with their new objective values. Basically, this means that a noisy sample of a solution is simply replaced by another noisy sample. It can be argued that the noise is actually not reduced and the original problem that the algorithm may be mislead by the noise is likely to remain. This might, in such a case, explain the weak results of the technique.

The worst results in the optimisations are achieved by the technique of fitness inheritance, especially in the real-world problems. Similar to the technique of dominance dependent lifetime, the poor results might be explained by an inadequate noise reduction approach. The technique of fitness inheritance uses a strategy in which the greater the standard deviation of a solution's parents (i.e. the larger their amount of noise), the greater the probability that the inherited values are accepted without resampling. Intuitively, the inverse is more logical, that is, the larger the noise the greater the need to perform resampling to reduce the noise.

User-defined parameters

All five noise handling techniques studied in this work suffer from arbitrary user-defined parameters. CDR requires the specification of two parameters: (*i*) confidence level, and (*ii*) maximum number of samplings. These parameters must be determined by the user based on the total number of simulations allocated, the desired level of accuracy of the optimisation, and the noise characteristics of the problem. The total number of simulations is usually known, but this might not be the case with the

desired level of accuracy and certainly not with the noise characteristics. It is therefore interesting to analyse whether or not the user must have a good understanding of how to set the confidence level and maximum number of samplings, that is, whether their configuration has a significant impact on the performance of the optimisation. To investigate this, we compared the results of a number of different parameter configurations on the ZDT1 benchmark problem with amount of noise 0.2 (Table 6.14). In the experiments, confidence levels ranging from 0.55 to 0.95 were tested in combination with three different settings of the maximum number of samplings.

Confidence level	Max. number of samplings (rank1-rank2-rank3)	Υ (minimise)	IGD (minimise)	S (maximise)
0.95	5-4-3	0.22	0.146	0.767
0.95	7-5-3	0.223	0.139	0.812
0.95	5-2-2	0.218	0.188	0.778
0.85	5-4-3	0.21	0.164	0.756
0.85	7-5-3	0.209	0.148	0.758
0.85	5-2-2	0.208	0.125	0.831
0.75	5-4-3	0.207	0.138	0.802
0.75	7-5-3	0.185	0.108	0.83
0.75	5-2-2	0.189	0.138	0.811
0.65	5-4-3	0.178	0.117	0.834
0.65	7-5-3	0.179	0.109	0.829
0.65	5-2-2	0.207	0.117	0.799
0.55	5-4-3	0.212	0.162	0.748
0.55	7-5-3	0.208	0.15	0.767
0.55	5-2-2	0.214	0.147	0.785

Table 6.14: Experiments on ZDT1 noise level 0.2.

As Table 6.14 illustrates, the configuration of the two parameters has some effect on the optimisation results. For the parameter "confidence level", it is quite clear that the extremes (i.e. level 0.95 and level 0.55) give the worst results, while levels in between give the best results (0.65 and 0.75). For the parameter "maximum number of samplings", a trend seems to be that doing fewer samplings is advantageous in combination with a high confidence level. An explanation of this might be that evolutionary algorithms tolerate, and can even be helped by, a small amount of

randomness in the evolutionary process. Excessive noise reduction may therefore lead to a waste of simulations.

The impact of parameter settings on performance is, however, relatively small and regardless of which configuration is considered, the CDR technique is still better than the other techniques (see Section 6.3.1). This indicates that the proposed technique does not rely upon a perfect parameter configuration, and that its performance is satisfactory even if the user is not sure of how to set the parameters. For maximum efficiency, however, trial-and-error in finding the optimal settings is needed.

Although the technique works well with a standard setting of the parameters, ideally there should be no user-defined parameters at all. Investigating how to get rid of the user-defined parameters, thereby making the use of the technique simpler, is an important topic for future research.

6.4 Parallel performance

This section presents the results from evaluating MOPSA-EA's parallel performance (Section 6.4.1) and also provides an analysis of the results (Section 6.4.2).

6.4.1 Results

The time needed by MOPSA-EA to optimise the Volvo Aero problem (including 400 simulation evaluations) with various numbers of processing nodes is presented in Table 6.15. As the results in Table 6.15 indicates, there is an obvious benefit from performing concurrent solution evaluations when simulations are computationally expensive; with one processing node the optimisation takes almost 50 hours to complete, while with 40 processing nodes it takes just over an hour. Based on the measured computational time, MOPSA-EA's speedup (cf. Equation 5.5) and efficiency (cf. Equation 5.6) were calculated, and these values are also presented in Table 6.15.

	Wall-clock time (hours:minutes:seconds)	Speedup	Efficiency
1 node	49:59:33	1	1
10 nodes	05:03:50	9.87	0.987
20 nodes	02:33:06	19.46	0.973
30 nodes	01:42:37	29.23	0.974
40 nodes	01:17:11	38.86	0.971

Table 6.15: Parallel performance of MOPSA-EA.

6.4.2 Analysis

Theoretically, the speedup of a parallel evolutionary algorithm can be linear at most, that is, the computational time can be reduced at most n times with n processing nodes (Hilbert et al., 2006). Super-linear speedup is not possible since the total execution time of a parallel evolutionary algorithm can never be less than that of a serial version of the algorithm (Cantú-Paz, 2001). MOPSA-EA achieves a near linear speedup, which is close to the theoretical maximum and indicates good scalability. Existing parallel multi-objective evolutionary algorithms report similar near-linear speedup (e.g. Kleeman et al., 2004; Jaimes and Coello Coello, 2005; Hilbert et al., 2006; Dehuri et al., 2006). The fact that no algorithm achieves linear speedup is not surprising, since, in practice, the communication overhead involved in parallel algorithms always reduces the speedup to below the theoretical maximum (Hilbert et al., 2006).

While speedup measures the gain from running an algorithm in parallel, efficiency measures the fraction of time that the processing nodes are working. The efficiency of MOPSA-EA is close to 1, which means that the processing nodes are well utilised. This is expected, since in the algorithm a new solution is immediately created when a slave node becomes free and the communication overhead is negligible in comparison to the time required for the simulation evaluations. The reported efficiency of existing parallel multi-objective evolutionary algorithms is generally worse than that of MOPSA-EA (e.g. Kleeman et al. 2004; Jaimes and Coello Coello 2005; Ciepiela et al. 2008), but due to parameter variations (e.g. simulation times,

communication schemes, and communication networks) comparisons of efficiency values are probably misleading.

The good speedup and efficiency of MOPSA-EA is not only thanks to the master-slave model, but the steady-state approach is also of significant importance. This becomes evident in considering the performance of the generation-based m-MAES (described in Section 5.4) when implemented with the master-slave model and applied to the Volvo Aero problem. The computational time, speedup, and efficiency of the parallelised version of m-MAES are presented in Table 6.16. Figure 6.6 shows a visualisation of m-MAES's speedup in comparison to that of MOPSA-EA. As the results in Table 6.16 indicate, although both m-MAES and MOPSA-EA use the master-slave model, there is a significant difference in their parallel performance. With more than 20 processing nodes, the speedup and efficiency of m-MAES can no longer be improved since the parallel performance of the generational approach is limited by the population size. This clearly shows the parallel benefits of the steady-state approach in comparison to the generation-based approach.

	Wall-clock time (hours:minutes:seconds)	Speedup	Efficiency
1 node	50:10:50	1	1
10 nodes	05:05:21	9.86	0.986
20 nodes	02:34:42	19.46	0.973
30 nodes	02:34:45	19.45	0.648
40 nodes	02:34:44	19.45	0.486

Table 6.16: Parallel performance of m-MAES.

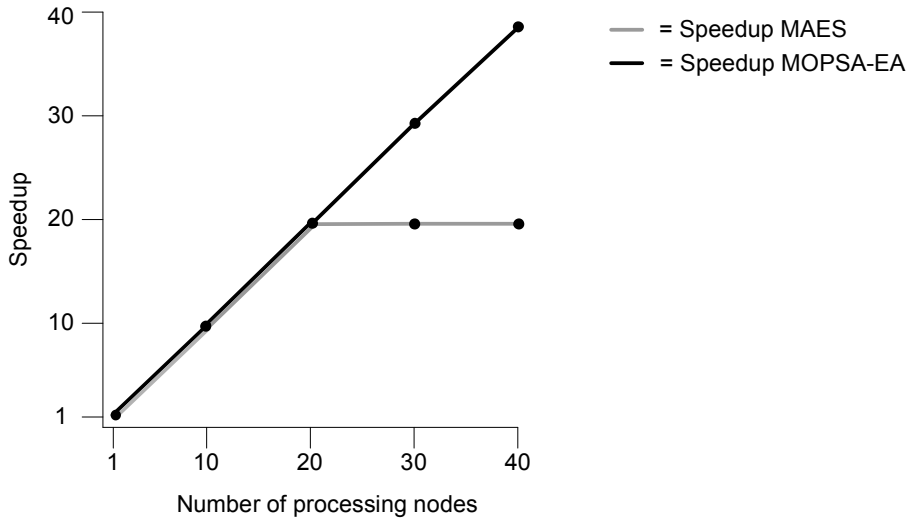


Figure 6.6: Speedup of m-MAES in comparison to MOPSA-EA.

In the evaluation of MOPSA-EA's parallel performance, a maximum of 40 processing nodes were tested. It is important to note that when the number of nodes is much larger and approaches the maximum number of simulation evaluations, some adjustments to the algorithm might be necessary in order to prevent it from turning into a random search. This is because in the algorithm, the master node initially generates random solutions and for maximum utilisation of slave nodes it continues with this as long as there are free slaves and no results have been communicated back (when the latter happens, new solutions are no longer created randomly but through evolutionary operators). If the number of slave nodes approaches the maximum number of simulation evaluations, there is a risk that a majority of the solutions will be random, since it usually takes at least a couple of minutes before the first results are received from the slave nodes - enough time for the master to create thousands of random solutions to most problems. This behaviour is obviously not desirable, but can easily be avoided, for example, by preventing the algorithm generating more than n solutions before a simulation result is received.

6.5 Summary

In this chapter, results from the evaluation of MOPSA-EA are presented along with an analysis of the algorithm. The main findings are:

- Both when noise is and when it is not considered in the optimisation, MOPSA-EA generates better solutions than existing, similar algorithms.
- The new technique to compensate for surrogate imprecision used in the algorithm is an important reason for the superior results of the algorithm.
- Another important reason for the superior results is the new noise handling technique used in the algorithm; this technique is able to effectively reduce noise and also compares favourably with other techniques for noise handling.
- The steady-state design of MOPSA-EA ensures that the speedup and the efficiency of the algorithm are close to the theoretical maximum.

Chapter 7

Conclusions

This chapter presents the overall conclusions of the research study and reflects on the hypothesis proposed in the beginning of the thesis (Section 7.1). It also outlines the main contributions of the research study (Section 7.2), and provides some guidance to the use of the new algorithm proposed in the study (Section 7.3). Possible future research is also discussed (Section 7.4).

7.1 Overall conclusions

In order to efficiently cope with the challenges imposed by real-world problems using simulation-based optimisation, a new evolutionary algorithm, called "multi-objective parallel surrogate-assisted evolutionary algorithm" (MOPSA-EA), is proposed in the thesis. The novelty of MOPSA-EA lies in its comprehensive approach of dealing with practical simulation-based optimisation problems; the algorithm takes into account multiple optimisation objectives, expensive simulation evaluations, and stochastic noise. The novelty of the algorithm also lies in the new techniques it adopts to compensate for surrogate imprecision and to reduce noise.

MOPSA-EA is evaluated on five benchmark problems and two complex real-world problems of manufacturing optimisation. The first real-world problem concerns the optimisation of a manufacturing cell for the production of aircraft and gas turbine

engine components at Volvo Aero. In order to improve processing, the overall utilisation of the cell needs to be increased and the number of overdue components must be minimised. The second real-world problem concerns the optimisation of a camshaft machining line at Volvo Cars Engine. In order to increase the throughput of the line and maintain minimum stock levels of the finished products, the scheduling of the line needs to be improved. The results from the optimisations show that MOPSA-EA finds better solutions to all the problems considered than existing, similar algorithms. The study identifies the new techniques for dealing with surrogate imprecision and noise used in the algorithm as key reasons for the good performance.

Consequently, the results confirm the hypothesis presented in the beginning of the thesis (Section 1.2.2):

"*An evolutionary algorithm that integrates techniques for multi-objective optimisation, parallelism, surrogate usage, and noise handling can achieve improved optimisation results when undertaking simulation-based optimisation of real-world problems*".

7.2 Contributions of this research

The main findings and scientific novelty of this research study are:

- **A new multi-objective evolutionary algorithm that efficiently tackles real-world problems using a simulation-based optimisation approach.**

  The new algorithm uses a Pareto approach for the handling of multiple optimisation objectives and searches for the set of best trade-off solutions. It supports a high degree of parallelism by adopting an asynchronous master-slave parallelisation model in combination with a steady-state design. A surrogate evaluation function is used in the algorithm to quickly identify promising candidate solutions and filter out poor ones. The imprecision associated with the surrogate is compensated for in order to avoid the propagation of inferior solutions.

Furthermore, the algorithm adopts a dynamic noise reduction procedure to tackle the problem of noise in simulations as a consequence of unpredictabilities in the real world.

- **A new technique to compensate for surrogate imprecision that is computationally cheap and easy to implement.**

  The new technique to compensate for surrogate imprecision is based on inheritance; the surrogate objective values of an offspring are modified by adding the weighted mean of the surrogate error values of the offspring's parents. Contrary to existing techniques that compensate for surrogate imprecision, this technique is parameterless, cheap in terms of computational cost, and scalable with respect to the number of objectives.

- **A new noise handling technique for multi-objective problems that effectively reduces noise by using a dynamic resampling scheme.**

  The new technique to deal with noise uses an iterative resampling procedure that efficiently reduces the noise by varying the number of samples used per solution based on the amount of noise in a local area of the search space. This dynamic strategy avoids wastful samplings when additional sampling is of little benefit, and promotes additional samplings when this is beneficial. Contrary to several existing techniques for noise handling, the new technique does not reduce noise for all solutions in the population, only for those participating in the evolutionary selection. The motivation for this approach is that noise is not harmful to every element of an evolutionary algorithm, only to those evolutionary processes that involve comparative selection. In other evolutionary operations, such as mating or mutation, noise is irrelevant.

From an industrial perspective, contributions to best practice in using simulation-based optimisation include:

- Demonstrating that the quality of solutions can be improved in simulation-based optimisation of complex real-world manufacturing systems.
- Showing it is possible to significantly reduce the optimisation time, thereby making simulation-based optimisation with a small time budget also feasible.
- Proving that fundamental characteristics of real-world problems, such as multiple objectives, high computational cost, and stochastic noise, can all be considered in the same optimisation algorithm.

7.3 User guidance

The proposed algorithm involves a number of user-defined parameters, and in the following default settings for these parameters are suggested. For the general parameters of the algorithm, the recommended settings are provided in Table 7.1.

Parameter	Setting
Population size	40
Number of offspring	40
Probability of mutation	0.1
Mutation step size	0.5
Crossover operator	Single-point
Crossover probability	0.8

Table 7.1: Recommended settings for general parameters of the algorithm.

As the surrogate evaluation function to incorporate in the algorithm, an artificial neural network is recommended with the settings provided in Table 7.2.

Parameter	Setting
Training set	50 most recent samples
Number of hidden layers	1
Training algorithm	Back-propagation
Activation function	Sigmoid
Learning rate	0.5
Cross validation	10-folded

Table 7.2: Recommended settings for artificial neural network.

Furthermore, the noise handling technique being part of the algorithm requires a number of user-defined parameters. Recommended settings for these are provided in Table 7.3.

	Confidence level	Max samplings
Pareto rank 1	0.75	5
Pareto rank 2	0.70	4
Pareto rank 3	0.65	3
Pareto rank 4	0.60	2
Pareto rank 5 and higher	0.55	2

Table 7.3: Recommended settings for noise handling technique.

7.4 Future work

Possible future research related to the research study is discussed below.

7.4.1 Robustness

A possible extension of the new algorithm is to consider robustness. The concept of robustness differs from noise in that the variations are not in the simulation (i.e. the objective function), but in the input parameters. If $f(x)$ is the objective function and δ represents the variance, then a noisy objective function is represented as $f'(x) = f(x) + \delta$, while a case of variations in the input parameters is represented as $f'(x) = f(x + \delta)$. In general, a solution is said to be robust if small changes in its parameters can be tolerated without a total loss of quality (Branke, 1998). In a buffer configuration problem, for example, a robust solution is one that still works satisfactorily even if a few slots of a buffer are broken. Robustness is important since in real-world problems it cannot usually be guaranteed that the exact parameters of a solution are actually implemented, but rather a solution close to the original one. While robustness has been studied in single-objective optimisation (see for example, Tsutsui and Gosh 1997; Branke 1998; Jin and Sendhoff 2003), the topic has only recently been considered in multi-objective optimisation (Deb and Gupta, 2006). In the context of MOPSA-

EA, a possibility might be to utilise the resamplings performed in its noise handling technique to bias the evolutionary process towards more robust solutions without any additional resamplings. This could be done by introducing small changes in the input parameters of a solution being resampled. A non-robust solution will then achieve poor results and be consequently punished in the evolutionary selection.

7.4.2 Output postprocessing of surrogate models

A drawback of surrogate models is that they are static, contrary to surrogate approximations, and do not improve during the optimisation (see Section 2.3.3). Even though new information about the underlying function of the simulation continuously becomes available when solutions are simulated, this information is not used to refine the surrogate model. One way to overcome this drawback might be to incorporate surrogate output post-processing. The idea is that it might be possible to train a surrogate approximation (e.g. an artificial neural network) to learn how the results from the surrogate model differ from the real simulation, and apply adjustments to the output so that the surrogate model can better match the output of the real simulation. In this way, the output post-processing can mask the simplifications made in the surrogate model and continuously increase the accuracy of the surrogate model during the optimisation.

7.4.3 Effects of combining a steady-state algorithm with the master-slave parallelisation model in multi-objective optimisation

In order to fit into the master-slave model, a steady-state algorithm must be designed to allow more than one individual to be created and simulated in parallel. A consequence of such a design is that solutions created in previous steps may not affect the creation of new solutions, because they have not yet been placed into the population. Furthermore, individuals might be returned in a different order than created, as

some nodes may finish their simulations faster than others. These consequences of combining a master-slave parallelisation model with a steady-state approach have previously been studied in single-objective evolutionary algorithms (e.g. Davison and Rasheed 1999), but do not seem to have received any attention in multi-objective steady-state evolutionary algorithms. It is particularly important to investigate the gain in time versus the degradation in performance when the number of slaves is increased.

7.4.4 Evaluation on problem with more than two objectives

In this study, optimisation problems of two objectives were considered in the evaluation of the proposed algorithm. To investigate how well the algorithm generalise to the case of more than two objectives, it is interesting to apply it also to m-objective problems (where $m > 2$). Problems of three objectives or more are, however, complex to analyse and performance comparisons of different algorithms on such problems are hard. Therefore, experiments on m-objective problems must be carefully prepared and supported by visualisation tools that facilitates the analysis of the results.

7.4.5 Benchmark problems for simulation-based optimisation

There is an increasing interest in using evolutionary algorithms for simulation-based optimisation of real-world problems, but no public platform is available to compare the performance of different algorithms applied to these problems. A set of standardised test problems is needed to make benchmarking possible. Such benchmark problems are also important with respect to replicability and verifiability of experiments. For relevant evaluations, the benchmark problems should have characteristics commonly found in real-world problems, such as high complexity (i.e. multimodality, non-separability, and high dimensionality) and multiple conflicting optimisation objectives (see Chapter 1). Ideally, the problems should be taken from the real world, preferably from different application domains. Furthermore, to mimic a real simulation scenario, the evaluation of solutions should be carried out using noisy, time

consuming black-box functions (i.e. functions that are not formally decidable). For the transparent comparison of experiments performed in different studies, executables of these functions should be accessible from a public repository.

References

Adamidis, P. (1998) Parallel evolutionary algorithms: A review, *Proceedings of 4th Hellenic-European Conference on Computer Mathematics and its Applications*, Athens, Greece.

Alba, E. and Luque, G. (2006) Evaluation of parallel metaheuristics, *Proceedings of the Workshop on Empirical Methods for the Analysis of Algorithms*, Reykjavik, Islandia, pp. 9–14.

Andersson, M., Persson, A., Grimm, H. and Ng, A. (2007) Simulation-based scheduling using a genetic algorithm with consideration to robustness: a real-world case study, *Proceedings of The 17th International Conference on Flexible Automation and Intelligent Manufacturing*, Philadelphia, USA, pp. 957–964.

April, J., Better, M., Glover, F. and Kelly, J. (2004) New advances for marrying simulation and optimization, *Proceedings of the 2004 Winter Simulation Conference*, Washington, DC, pp. 80–86.

April, J., Glover, F., Kelly, J. and Laguna, M. (2001) Simulation/optimization using "real-world" applications, *Proceedings of the 2001 Winter Simulation Conference*, Arlington, VA, pp. 134–138.

Arnold, D. and Beyer, H. (2002) *Noisy Local Optimization with Evolution Strategies*, Kluwer Academic Publishers, Norwell, MA ISBN: 1402071051.

Babbar, M., Lakshmikantha, A. and Goldberg, D. E. (2003) A modified NSGA-II

to solve noisy multiobjective problems, *Proceedings of Genetic and Evolutionary Computation Conference*, Vol. 2723 of *Lecture Notes in Computer Handling Uncertainty in Indicator-Based Multiobjective Optimization*, Springer Verlag, Chicago, Illinois, USA, pp. 21–27.

Basseur, M. and Zitzler, E. (2006) Handling uncertainty in indicator-based multiobjective optimization, *International Journal of Computational Intelligence Research* **2**(3): 255–272.

Baxter, B. (1992) *The Interpolation Theory of Radial Basis Functions*, PhD thesis, Trinity College, Cambridge University.

Büche, D., Schraudolph, N. and Koumoutsakos, P. (2005) Accelerating evolutionary algorithms with gaussian process fitness function models, *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews* **35**(2): 183–194.

Büche, D., Stoll, P., Dornberger, R. and Koumoutsakos, P. (2002) An evolutionary algorithm for multi-objective optimization of combustion processes, *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* **32**(4): 460–473.

Bäck, T., Fogel, D. and Michalewicz, Z. (eds) (1997) *Handbook of Evolutionary Computation*, Oxford University Press.

Beyer, H. (2000) Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice, *Computer Methods in Applied Mechanics and Engineering* **186**(2-4): 239–267.

Beyer, H. and Schwefel, H. (2002) Evolution strategies - a comprehensive introduction, *Natural Computing* **1**(1): 3–52 Springer-Verlag Netherlands.

Blanning, R. (1975) The construction and implementation of metamodels, *Simulation* pp. 177–184.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G. and Weglarz, J. (2001) *Scheduling Computer and Manufacturing Processes*, 2nd edn, Springer Verlag, Berlin.

Boesel, J., Bowden, R., Glover, F., Kelly, J. and Westwig, E. (2001) Future of simulation optimization, *Proceedings of the 2001 Winter Simulation Conference*, Arlington, VA, pp. 1466–1470.

Bradstreet, L., While, L. and Barone, L. (2008) A fast incremental hypervolume algorithm, *IEEE Transactions on Evolutionary Computation* **12**(6): 714–723.

Branke, J. (1998) Creating robust solutions by means of an evolutionary algorithm, *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, Vol. 1498 of *Lecture Notes In Computer Science*, Springer London, Amsterdam, The Netherlands, pp. 119–128.

Branke, J., Meisel, S. and Schmidt, C. (2007) Simulated annealing in the presence of noise, *Journal of Heuristics* **14**(6): 627–654.

Branke, J. and Schmidt, C. (2003) Selection in the presence of noise, *in* E. C.-P. et.al. (ed.), *Proceedings of Genetic and Evolutionary Computation Conference*, number LNCS 2273, Springer-Verlag Berlin Heidelberg, Chicago, Illinois, pp. 766–777.

Branke, J., Schmidt, C. and Schmeck, H. (2001) Efficient fitness estimation in noisy environments, *in* L. S. et al. (ed.), *Proceedings of Genetic and Evolutionary Computation Conference*, Morgon Kaufmann, San Francisco, California, pp. 243–250.

Bui, L., Abbass, H. and Essam, D. (2005) Fitness inheritance for noisy evolutionary multi-objective optimization, *Proceedings of Genetic and Evolutionary Computation Conference*, Washington, DC, USA, pp. 779–785.

Bui, L., Abbass, H., Essam, D. and Green, D. (2004) Performance analysis of evolutionary multi-objective optimization methods in noisy environments,

Proceedings of the 8th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Queensland, Australia, pp. 29–39.

Cantú-Paz, E. (2000) *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic Publishers.

Cantú-Paz, E. (2001) Migration policies, selection pressure, and parallel evolutionary algorithms, *Journal of Heuristics archive* **7**(4): 311–334.

Chafekar, D., Shi, L., Rasheed, K. and Xuan, J. (2005) Constrained multi-objective ga optimization using reduced models, *IEEE Transactions on Systems, Man and Cybernetics* **35**(2): 261–265.

Chafekar, D., Xuan, J. and Rasheed, K. (2003) Constrained multi-objective optimization using steady state genetic algorithms, *Proceedings of The Genetic and Evolutionary Computation Conference*, Chicago, Illinois, pp. 813–824.

Chen, J.-H., Goldberg, D., Ho, S.-Y. and Sastry, K. (2002) Fitness inheritance in multi-objective optimization, *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, pp. 319–326.

Chen, T., Chen, H. and Liu, R. (1995) Approximation capability in C(Rn) by multilayer feedforward networks and related problems, *IEEE Transactions on Neural Networks* **6**(1): 25–30.

Ciepiela, E., Kocot, J., Siwik, L. and Drezewski, R. (2008) Hierarchical approach to evolutionary multi-objective optimization, *International Conference on Computational Science*, Krakow, Poland, pp. 740–749.

Coello Coello, C., Lamont, G. and Veldhuizen, D. V. (2007) *Evolutionary Algorithms for Solving Multi-Objective Problems*, Genetic and Evolutionary Computation, 2nd edn, Kluwer Academic Publishers.

Coello Coello, C. and Reyes-Sierra, M. (2004) A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm, *Proceedings of Third Mexican International Conference on Artificial Intelligence*, Mexico City, Mexico, pp. 688–697.

Cormen, T.H., L. C. R. R. and Stein, C. (2001) *Introduction to Algorithms. . USA:* ., 2nd edn, MIT Press.

Davison, B. and Rasheed, K. (1999) Effect of global parallelism on a steady state genetic algorithm, *Proceedings of Genetic and Evolutionary Computation Conference*, Orlando, Florida, pp. 167–170.

de Toro Negro, F., Ortega, J., Ros, E., Mota, S., Paechter, B. and Martin, J. (2004) Psfga: Parallel processing and evolutionary computation for multi-objective optimization, *Parallel Computing* **30**(5-6): 721–739.

Deb, K. (1999) Multi-objective genetic algorithms: Problem difficulties and construction of test functions, *Evolutionary Computation* **7**: 205–230.

Deb, K. (2004) *Multi-Objective Optimization using Evolutionary Algorithms*, second edn, John Wiley & Sons Ltd.

Deb, K. and Goel, T. (2001) Controlled elitist non-dominated sorting genetic algorithms for better convergence, *First International Conference on Evolutionary Multi-Criterion Optimisation*, Zurich, pp. 67–81.

Deb, K. and Gupta, H. (2006) Introducing robustness in multi-objective optimization, *Evolutionary Computation* **14**(4): 463–494.

Deb, K., Mohan, M. and Mishra, S. (2003) A fast multi-objective evolutionary algorithm for finding well-spread pareto-optimal solutions, *KanGAL Report 2003002*, Indian Institute of Technology Kanpur, India.

Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2000) A fast and elitist multi-objective genetic algorithm NSGA-II, *KanGAL Report 2000001*, Indian Institute of Technology Kanpur, India.

Deb, K., Thiele, L., Laumanns, M. and Zitzler, E. (2002) Scalable multi-objective optimization test problems, *Proceedings of Congress on Evolutionary Computation*, IEEE Press, pp. 825–830.

Deb, K., Thiele, L., Laumanns, M. and Zitzler, E. (2005) *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*, Springer, USA, chapter Scalable Test Problems for Evolutionary Multiobjective Optimization, pp. 105–145.

Dehuri, S., Jagadev, A. K., Ghosh, A. and Mall, R. (2006) Multi-objective genetic algorithm for association rule mining using a homogeneous dedicated cluster of workstations, *American Journal of Applied Sciences* **3**(11): 2086–2095.

Eiben, A. E. and Schippers, C. A. (1998) On evolutionary exploration and exploitation, *Fundamenta Informaticae* **35**(1-4): 35–50.

El-Beltagy, M., Mohammed, A., Keane, A. and Andy, J. (2001) Evolutionary optimization for computationally expensive problems using gaussian processes, *Proceedings of the International Conference on Artificial Intelligence*, CSREA Press, Las Vegas, Nevada, pp. 708–714.

El-Beltagy, M., Nair, P. and Keane, A. (1999) Metamodeling techniques for evolutionary optimization of expensive problems: Promises and limitations, *Proceedings of Genetic and Evolutionary Computation Conference*, Orlando, Florida, pp. 196–203.

Emmerich, M. (2005) *Single- and Multi-Objective Evolutionary Design Optimization Assisted by Gaussian Random Field Metamodels*, PhD thesis, University of Dortmund.

Emmerich, M., Beume, N. and Naujoks, B. (2005) An EMO algorithm using the hypervolume measure as selection criterion, *Proceedings of Evolutionary Multi-Criterion Optimization*, Vol. 3410 of *Lecture Notes in Computer Science*, Publisher Springer Berlin / Heidelberg, Guanajuato, Mexico.

Emmerich, M., Giannakoglou, K. and Naujoks, B. (2006) Single- and multi-objective evolutionary optimization assisted by gaussian random field metamodels, *IEEE-TEC Special Issue on Evolutionary Computation in the presence of uncertainty* **10**(4): 421–439.

Emmerich, M., Giotis, A., Özdemir, M., Bäck, T. and Giannakoglou, K. (2002) Metamodelassisted evolution strategies, *in* J. M. G. et al. (ed.), *Proceedings of the International Conference on Parallel Problem Solving from Nature*, Springer-Verlag, Berlin Heidelberg, Granada, Spain, pp. 361–370.

Emmerich, M. and Naujoks, B. (2004) Metamodel-assisted multiobjective optimisation strategies and their application in airfoil design, *in* I. Parmee (ed.), *Proceedings of the Fifth International Conference on Adaptive Design and Manufacture*, Springer Berlin, Bristol, UK, pp. 249–260.

Eskandari, H., Rabelo, L. and Mollaghasemi, M. (2005) Multiobjective simulation optimization using an enhanced genetic algorithm, *Proceedings of the 2005 Winter Simulation Conference*, Orlando, Florida, pp. 833–841.

Evans, G., Stuckman, B. and Mollaghasemi, M. (1991) Multicriteria optimization of simulation models, *Proceedings of 1991 Winter Simulation Conference*, Phoenix, Arizona, pp. 894–900.

Fieldsend, J. and Everson, R. (2005) Multi-objective optimisation in the presence of uncertainty, *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, Vol. 1, Edinburgh, UK, pp. 243–250.

Fishman, G. S. (2001) *Discrete-Event Simulation: Modeling, Programming, and Analysis*, Springer-Verlag, Berlin.

Fitzpatric, J. and Grefenstette, J. (1988) Genetic algorithms in noisy environments, *Machine Learning* **3**: 101–120.

Fonseca, C. and Fleming, P. (1993) Genetic algorithms for multiobjective optimization: Formulation, discussion, and generalization, *Proceedings of the Fifth International Conference on Genetic Algorithms*, Urbana-Champaign, Illinois, pp. 416–423.

Giannakoglou, K. (2002) Design of optimal aerodynamic shapes using stochastic optimization methods and computational intelligence, *Progress in Aerospace Sciences* **38**(1): 43–76.

Giotis, A., J., K. G. and Periaux (2000) A reduced-cost multi-objective optimization method based on the pareto front technique, neural networks and pvm, *Proceedings of European Congress on Computational Methods in Applied Sciences and Engineering*, Barcelona , Spain.

Giunta, A. A. and Watson, L. T. (1998) A comparison of approximation modeling techniques: Polynomial versus interpolating models, *Proceedings of the Seventh AIAA/USAF/NASA/ISSMO symposium on multidisciplinary analysis and optimization*, St. Louis, Missouri.

Goh, C. and Tan, K. (2006) Noise handling in evolutionary multi-objective optimization, *Proceedings of IEEE Congress on Evolutionary Computation*, Vancouver, BC, Canada, pp. 1354–1361.

Goldberg, D. (1989) *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley Publishing Co, Boston, Massachusetts.

Hilbert, R., Janiga, G., Baron, R. and Thévenin, D. (2006) Multi-objective shape

optimization of a heat exchanger using parallel genetic algorithms, *International Journal of Heat and Mass Transfer* **49**(15-16): 2567–2577.

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

Horn, J., Nafploitis, N. and Goldberg, D. (1994) A niched pareto genetic algorithm for multi-objective optimisation, *Proceedings of the First IEEE Conference on Evolutionary Computation*, Orlando, Florida, pp. 82–87.

Hüsken, M., Jin, Y. and Sendhoff, B. (2005) Structure optimization of neural networks for evolutionary design optimization, *Soft Computing - A Fusion of Foundations, Methodologies and Applications* **9**(1): 21– 28 Springer-Verlag Berlin, Heidelberg.

Huband, S., Barone, L., While, L. and Hingston, P. (2005) A scalable multi-objective test problem toolkit, *Proceedings of 3rd International Conference on Evolutionary Multi-Criterion Optimization*, Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 280–295.

Huband, S., Hingston, P., While, L. and Barone, L. (2003) An evolution strategy with probabilistic mutation for multi-objective optimisation, *Proceedings of the 2003 Congress on Evolutionary Computation*, Vol. 4, Canberra, Australia, pp. 2284–2291.

Hughes, E. (2001) Evolutionary multi-objective ranking with uncertainty and noise, *Proceedings of First International Conference on Evolutionary Multi-Criterion Optimization*, Springer Berlin-Heidelberg, Zurich, Switzerland, pp. 329–343.

Igel, C., Hansen, N. and Roth, S. (2007) Covariance matrix adaptation for multi-objective optimization, *Evolutionary Computation* **15**(1): 1–28.

Jaimes, A. and Coello Coello, C. (2005) Mrmoga: Parallel evolutionary multiobjective optimization using multiple resolutions, *Proceedings of IEEE Congress on Evolutionary Computation*, IEEE Press, Edinburgh, Scotland, pp. 2294–2301.

Jensen, M. T. (2003) Reducing the run-time complexity of multiobjective eas: The NSGA-II and other algorithms, *IEEE Transactions on Evolutionary Computation* **7**(5): 503–515.

Jin, R., Chen, W. and Simpson, T. (2001) Comparative studies of metamodelling techniques under multiple modelling criteria, *Structural Multidisciplinary Optimization* **23**: 1–13 Springer-Verlag.

Jin, Y. (2005) A comprehensive survey of fitness approximation in evolutionary computation, *Soft Computing* **9**: 3–12 Springer-Verlag.

Jin, Y. and Branke, J. (2005) Evolutionary optimization in uncertain environments - a survey, *IEEE Transactions on Evolutionary Computation* **9**(3): 303–317.

Jin, Y., Hüsken, M. and Sendhoff, B. (2003) Quality measures for approximate models in evolutionary computation, *Proceedings of Genetic and Evolutionary Computation Conference*, Chicago, Illinois, pp. 170–173.

Jin, Y., Olhofer, M. and Sendhoof, B. (2002) A framework for evolutionary optimization with approximate fitness functions, *IEEE Transactions on Evolutionary Computation* **6**(5): 481–494.

Jin, Y. and Sendhoff, B. (2003) Trade-off between performance and robustness: An evolutionary multiobjective approach, *Proceedings of Second International Conference on Evolutionary Multi-Criterion Optimization*, Faro, Portugal, pp. 237–251.

Jones, J. and Soule, T. (2006) Comparing genetic robustness in generational vs. steady state evolutionary algorithms, *Proceedings of Genetic and Evolutionary Computation Conference*, Seattle, WA, USA, pp. 143–150.

Karakasis, M. and Giannakoglou, K. (2004) On the use of surrogate evaluation models

in multi-objective evolutionary algorithms, *Proceedings of European Congress on Computational Methods in Applied Sciences and Engineering*, Jyväskylä, Finland.

Karakasis, M. and Giannakoglou, K. (2005) Metamodel-assisted multi-objective evolutionary optimization, *Proceedings of Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems*, Munich, Germany.

Karp, A. and Flatt, H. (1990) Measuring parallel processor performance, *Communications of the ACM* **33**(5): 539–543.

Kim, I. and Weck, O. (2006) Adaptive weighted sum method for multiobjective optimization: a new method for pareto front generation, *Structural and Multidisciplinary Optimization* **31**(2): 105–116.

Kleeman, M., Day, R. and Lamont, G. (2004) Analysis of a parallel moea solving the multi-objective quadratic assignment problem, *Proceedings of Genetic and Evolutionary Computation Conference*, Springer Berlin Heidelberg, Seattle, Washington, pp. 402–403.

Knowles, J. (2005) A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers, *Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, Wroclaw, Poland, pp. 552–557.

Knowles, J. and Corne, D. (2000) Approximating the non-dominated front using the pareto archived evolution strategy, *Evolutionary Computation Journal* **8**(2): 149–172.

Krige, D. (1966) A study of gold and uranium distribution patterns in the klerksdorp gold field, *Geoexploration* **4**(1): 43–53.

Lacksonen, T. (2001) Empirical comparison of search algorithms for discrete event simulation, *Computers and Industrial Engineering* **40**(1-2): 133–148.

Ólafsson, S. and Kim, J. (2002) Simulation optimization, *Proceedings of the 2002 Winter Simulation Conference*, San Diego, Californien, pp. 79–85.

Laguna, M. and Marti, R. (2002) Neural network prediction in a system for optimizing simulations, *IEE Transactions* **34**: 273–282.

Laguna, M. and Marti, R. (2003) *Optimization Software Class Libraries*, Kluwer Academic Publishers, Boston, chapter The OptQuest Callable Library, pp. 193–218.

Laumanns, M., Thiele, L., Deb, K. and Zitzler, E. (2002) Combining convergence and diversity in evolutionary multiobjective optimization, *Evolutionary Computation* **10**(3): 263–282.

Law, A. M. and Kelton, D. (2000) *Simulation Modeling and Analysis*, third edn, Mc Graw Hill.

Lee, L., Chew, E., Teng, S. and Chen, Y. (2008) Multi-objective simulation-based evolutionary algorithm for an aircraft spare parts allocation problem, *European Journal of Operational Research* **189**(2): 476–491.

Lim, D., Ong, Y. and Lee, B. (2005) Inverse multi-objective robust evolutionary design optimization in the presence of uncertainty, *Proceedings of the 2005 Workshops on Genetic and Evolutionary Computation*, Washington, D.C., pp. 55–62.

Lobo, F., Lima, F. and Michalewicz, Z. (eds) (2007) *Parameter Setting in Evolutionary Algorithms*, Vol. 54 of *Studies in Computational Intelligence*, Springer Verlag.

Lozano, M., Herrera, F. and Cano, J. (2008) Replacement strategies to preserve useful diversity in steady-state genetic algorithms, *Journal of Information Sciences* **178**(23): 4421–4433.

Mehnen, J., Michelitsch, T., Bartz-Beielstein, T. and Henkenjohann, N. (2004) Systematic analyses of multi-objective evolutionary algorithms applied to real-world problems using statistical design of experiments, *Proceedings of 4th CIRP*

International Seminar on Intelligent Computation in Engineering, Sorrento, Italy, pp. 171–178.

Mehrotra, K., Mohan, C. and Ranka, S. (1996) *Elements of Artificial Neural Networks*, MIT Press ISBN 0-262-13328-8.

Miller, B. and Goldberg, D. (1995) Genetic algorithms, tournament selection, and the effects of noise, *Complex System* **9**(3): 193–212.

Miller, B. and Goldberg, D. (1996) Genetic algorithms, selection schemes, and the varying effect of noise, *Evolutionary Computation* **4**(2): 113–131.

Mostaghim, S., Branke, J., Lewis, A. and Schmeck, H. (2008) Parallel multi-objective optimization using master-slave model on heterogeneous resources, *Proceedings of IEEE Congress on Evolutionary Computation*, Hong Kong, China.

Mumford, C. (2004) Simple population replacement strategies for a steady-state multi-objective evolutionary algorithm, *Proceedings of Genetic and Evolutionary Computation Conference*, Springer Berlin Heidelberg, Seattle, Washington.

Nain, P. and Deb, K. (2005) A multi-objective optimization procedure with successive approximate models, *Technical Report KanGAL Report No. 2005002*, Indian Institute of Technology, India.

Naujoks, B., Beume, N. and Emmerich, M. (2005) Multi-objective optimisation using s-metric selection: application to three-dimensional solution spaces, *Proceedings of The 2005 IEEE Congress on Evolutionary Computation*, Vol. 2, Edinburgh, UK, pp. 1282–1289.

Nebro, A., Durillo, J., Luna, F., Dorronsoro, B. and Alba, E. (2007) Design issues in a multiobjective cellular genetic algorithm, *Proceeding of the Fourth International Conference on Evolutionary Multi-Criterion Optimization*, Vol. 4403 of *Lecture Notes in Computer Science*, Springer Verlag, Matsushima Sendai, Japan.

Ng, A., Grimm, H., Lezama, T., Persson, A., Andersson, M. and Jägstam, M. (2007) Web services for metamodel-assisted parallel simulation optimization, *Proceedings of The IAENG International Conference on Internet Computing and Web Services*, Hong Kong, pp. 879–885.

Ochoa, G., Harvey, I. and Buxton, H. (1999) On recombination and optimal mutation rates, *Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, Florida, pp. 13–17.

Ong, Y., Nair, P., Keane, A. and Wong, K. (2004) Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems, *Knowledge Incorporation in Evolutionary Computation* pp. 307–332.

Persson, A., Grimm, H., Ng, A., Lezema, T., Ekberg, J., Falk, S. and Stablum, P. (2006) Simulation-based multi-objective optimization of a real-world operation scheduling problem, *Proceedings of the 2006 Winter Simulation Conference*, Monterey, California, pp. 1757–1764.

Pietro, A. D., While, L. and Barone, L. (2004) Applying evolutionary algorithms to problems with noisy, time-consuming fitness functions, *Proceedings of IEEE Congress on Evolutionary Computation*, Vol. 2, Portland, Oregon, pp. 1254– 1261.

Pilato, C., Palermo, G., Tumeo, A., Ferrandi, F., Sciuto, D. and Lanzi, P. (2007) Fitness inheritance in evolutionary and multi-objective high-level synthesis, *Proceedings of IEEE Congress on Evolutionary Computation*, Singapore, pp. 3459–3466.

Queipo, N. V., Haftka, R. T., Shyy, W., Goel, T., Vaidyanathan, R. and Tucker, K. (2005) Surrogate-based analysis and optimization, *Progress in Aerospace Sciences* **41**: 1–28.

Ratle, A. (1999) Optimal sampling strategies for learning a fitness model, *Proceedings of 1999 Congress Evolutionary Computation*, Washington, DC, pp. 2078–2085.

Rechenberg, I. (1965) Cybernetic solution path of an experimental problem, *Technical Report 1122*, Royal Aircraft Establishment, Library Translation, Farnborough, Hants, UK.

Reyes-Sierra, M. and Coello Coello, C. (2005) Fitness inheritance in multi-objective particle swarm optimization, *Proceedings of IEEE Swarm Intelligence Symposium*, Pasadena, California, pp. 116–123.

Sacks, J., W.J., Mitchell, T. and Wynn, H. (1989) Design and analysis of computer experiments, *Statistical Science* **4**: 409–435.

Schaffer, D. (1984) *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*, PhD thesis, Vanderbilt University, Nashville, TN.

Shir, O., Emmerich, M., Bäck, T. and Vrakking, M. (2007) The application of evolutionary multi-criteria optimization to dynamic molecular alignment, *Proceedings of IEEE Congress on Evolutionary Computation*, IEEE Press, Singapore.

Simpson, T., Mauery, T., Korte, J. and Mistree, F. (1998) Comparison of response surface and kriging models for multidisciplinary design optimization, *Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Vol. 1, St. Louis, Missouri, pp. 381–391.

Smith, R., Dike, B. and Stegmann, S. (1995) Fitness inheritance in genetic algorithms, *Proceedings of the 1995 ACM Symposium on Applied Computing*, Nashville, Tennessee.

Srinivas, N. and Deb, K. (1995) Multiobjective optimization using nondominated sorting in genetic algorithms, *Evolutionary Computation* **2**(3): 221–248.

Srinivasan, D. and Rachmawati, L. (2006) An efficient multi-objective evolutionary algorithm with steady-state replacement model, *Proceedings of Genetic and*

Evolutionary Computation Conference, number ISBN:1-59593-186-4, Seattle, Washington, pp. 715–722.

Streichert, F., Ulmer, H. and Zell, A. (2005) Parallelization of multi-objective evolutionary algorithms using clustering algorithms, *in* C. Coello-Coello, A. Hernández and E. Zitzler (eds), *Proceedings of Third International Conference on Evolutionary Multi-Criterion Optimization*, Vol. 3410 of *Lecture Notes in Computer Science*, Springer, Guanajuato, Mexico, pp. 92–107.

Swisher, J., Hyden, P., Jacobson, S. and Schruben, L. (2000) A survey of simulation optimization techniques and procedures, *Proceedings of the 2000 Winter Simulation Conference*, Orlando, Florida, pp. 119–129.

Tan, K. C. and Goh, C. K. (2008) Handling uncertainties in evolutionary multi-objective optimization, *Proceedings of IEEE World Congress on Computational Intelligence*, Vol. 5050 of *Lecture Notes in Computer Science*, Springer, Hong Kong, China, pp. 262–292.

Teich, J. (2001) Pareto-front exploration with uncertain objectives, *Proceedings of First International Conference on Evolutionary Multi-Criterion Optimization*, Springer Berlin Heidelberg, Zurich, Switzerland, pp. 314–328.

Tran, K. (2006) *An Improved Multi-Objective Evolutionary Algorithm with Adaptable Parameters*, PhD thesis, Graduate School of Computer and Information Sciences, Nova Southeastern University.

Tsao, C. (2007) Comparison between response surface methodology and radial basis function network for core-center drill in drilling composite materials, *International Journal of Advanced Manufacturing Technology* (11-12).

Tsutsui, S. and Gosh, A. (1997) Genetic algorithms with a robust searching scheme, *IEEE Transactions on Evolutionary Computation* **1**(3): 201–219.

Ulmer, H., Streichert, F. and Zell, A. (2003a) Evolution strategies assisted by gaussian processes with improved pre-selection criterion, *Proceedings of IEEE Congress on Evolutionary Computation (CEC03)* pp. 692–699 Canberra, Australia, December 8-12.

Ulmer, H., Streichert, F. and Zell, A. (2003b) Optimization by gaussian processes assisted evolution strategies, *Selected Papers of the International Conference on Operations Research* pp. 434–442 Heidelberg, Germany, 3-5 September.

Vavak, F. and Fogarty, T. (1996) A comparative study of steady state and generational genetic algorithms, *Selected Papers from AISB Workshop on Evolutionary Computing*, Brighton, UK, pp. 297–304.

Veldhuizen, D. V., Zydallis, J. and Lamont, G. (2002) Issues in parallelizing multiobjective evolutionary algorithms for real world applications, *Proceedings of the 2002 ACM symposium on Applied Computing*, Madrid, Spain.

Veldhuizen, D., Zydallys, J. and Lamont, G. (2003) Considerations in engineering parallel multiobjective evolutionary algorithms, *IEEE Transactions on Evolutionary Computation* **7**(2): 144–173.

Villaseñor, D., Morales-Menéndez, R., Rodríguez, C. and Alique, J. R. (2006) Neural networks and statistical based models for surface roughness prediction, *Proceedings of the IASTED International Conference on Modeling, Identification, and Control*, Lanzarote, Spain, pp. 326–331.

Voutchkov, I. and Keane, A. J. (2006) Multiobjective optimization using surrogates, *Proceedings of International Conference on Adaptive Computing in Design and Manufacture*, Bristol, U.K., pp. 167–175.

Xiong, S. and Li, F. (2003) Parallel strength pareto multiobjective evolutionary algorithm, *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies*, Sichuan, China, pp. 681–683.

Yang, Y., Jang, B., Yeun, Y. and Ruy, W. (2002) Managing approximation models in multiobjective optimization, *Structural and Multidisciplinary Optimization* **24**: 141–156.

Zitzler, E. (1999) *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*, PhD thesis, Eidgenössische Technische Hochschule, Swiss Federal Institute of Technology, Zurich.

Zitzler, E., Brockho, D. and Thiele, L. (2007) The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration, *Proceedings of the Fourth International Conference on Evolutionary Multi-Criterion Optimization*, Vol. 4403 of *Lecture Notes in Computer Science*, Springer, pp. 862–876.

Zitzler, E., Deb, K. and Thiele, L. (2000) Comparison of multiobjective evolutionary algorithms: Empirical results, *Evolutionary Computation* **8**(2): 173–195.

Zitzler, E., Laumanns, M. and Thiele, L. (2001) SPEA2: Improving the strength pareto evolutionary algorithm, *Technical Report 103*, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology.

Zitzler, E. and Thiele, L. (1998) An evolutionary algorithm for multiobjective optimization: The strength pareto approach, *Technical Report 43*, Computer Engineering and Communication Networks Lab, Swiss Federal Institute of Technology.

Appendix A
Characteristics of real-world problems

Real-world problems are often multimodal, non-separable and of high dimensionality. *Multimodality* refers to search spaces with one or more *local optimum,* i.e. a solution that is optimal within a neighbouring set of solutions, in contrast to a *global optimum* which is the optimal solution among the complete set of solutions (Figure A.1). *Non-separability* means that the optimisation depends on the interaction of two or more input parameters, as opposed to a *separable* problem that can be optimised by considering each parameter independently of each other. The problem of buffer capacity optimisation introduced in Chapter 1, for example, is non-separable considering that the configuration of one buffer influences the production flow and thereby also the optimal configurations of all other buffers. *Dimensionality* refers to the number of input parameters.

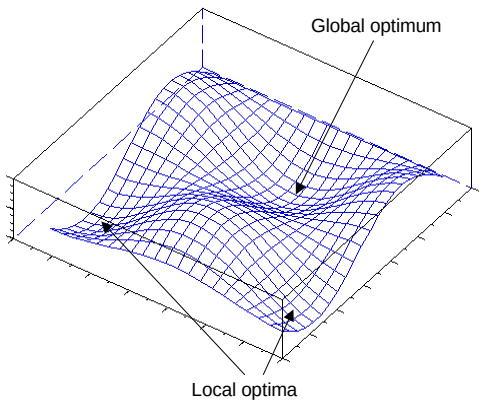


Figure A.1: Multimodal search space.

Appendix B
Basics of evolutionary algorithms

The basic idea behind evolutionary algorithms is to use computational models of evolutionary processes in the design and implementation of problem solving applications. Based on Darwin's theory of "survival of the fittest", candidate solutions to a problem are iteratively refined. Generally, an evolutionary algorithm consists of a genetic representation of solutions, a population-based solution approach, an iterative evolutionary process, and a guided random search. The *genetic representation* defines an individual solution as a set of genes. Returning to the manufacturing system example described in Chapter 1, a gene may correspond to a specific buffer, and the value of the gene to the size of the buffer, as illustrated in Figure B.1 for a system of four buffers.

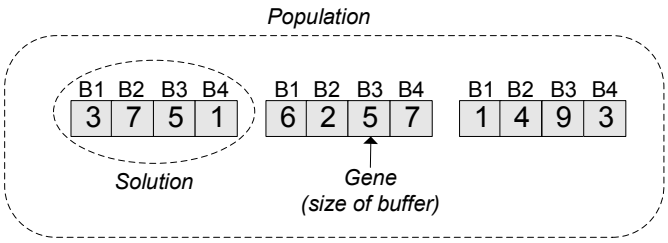


Figure B.1: Genetic representation of solutions.

In evolving a population of solutions, evolutionary algorithms apply biologically inspired operations for selection, crossover and mutation. The operators are applied in a loop, and an iteration of the loop is called a *generation*. In Algorithm 5, the basic steps involved in this evolutionary process are presented in the pseudo code .

Algorithm 5 Basic steps of an evolutionary algorithm.

```
Initialise population
Evaluate the fitness of solutions in the population
repeat
    Select solutions to reproduce
    Form a new generation of population through crossover and mutation
    Evaluate the new solutions
until terminating condition
```

The solutions in the initial population are usually generated randomly. During each generation, a proportion of the solutions in the population is selected to breed offspring for the next generation of the population. Either a complete population is created immediately and used to replace the old population (*generational approach*), or one solution at a time is created and used to replace one of the solutions in the population (*steady-state approach*). Solutions are selected based on their *fitness*, representing a quantification of their optimality. For example, in the buffer optimisation problem, fitness may be the weighted sum of throughput and work-in-process. Typically, solutions with high fitness have a higher probability o be selected, but to prevent premature convergence, it is common that a small proportion of solutions with worse fitness are also selected. From the solutions selected, new solutions are created to form the next generation of the population. For to creation of each new solution, two parent solutions are chosen and through mating (called *crossover*) an offspring is produced (Figure B.2).

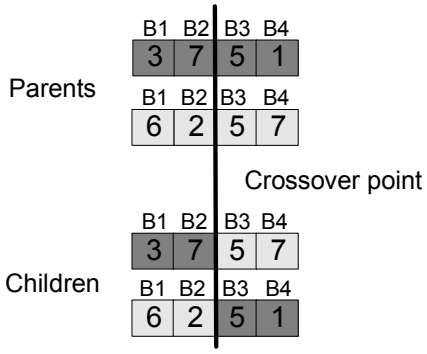


Figure B.2: Crossover.

Occasionally, the new solution can undergo a small mutation in order to keep the diversity of the population large and avoid local minima. A *mutation* usually involves changing an arbitrary part of a solution with a certain probability, for example slightly changing the size of a buffer in Figure B.3. When the new population is formed, the average fitness will have generally increased. The process of evolving generations continues until a user-defined termination criterion has been fulfilled, for example that the best solutions in the last n generations have not changed.

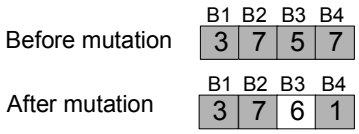


Figure B.3: Mutation.

Two well-defined evolutionary algorithms have served as the basis for much of the activity in the field: *evolution strategies* and *genetic algorithms*, which are described below.

Evolution strategies Evolution strategies were founded in the middle of the 1960s (Rechenberg, 1965). In an evolution strategy, the selection of parents for breeding offspring is random-based and independent of the parents' fitness values. Offspring are mutated by modifying them with a normally distributed random value, where the standard deviation of the normal distribution is usually self-adaptive (i.e. automatically learned during the optimisation). Different variants of evolution strategies exist (Beyer and Schwefel (2002)):

(1 + 1) In this strategy, which is the simplest form of an ES, selection takes place between one parent and one offspring (also called two-member evolution strategies exist).

(μ + 1) In this strategy, the population consists of μ and two parents are chosen to breed one offspring (also called steady-state evolution strategies exist).

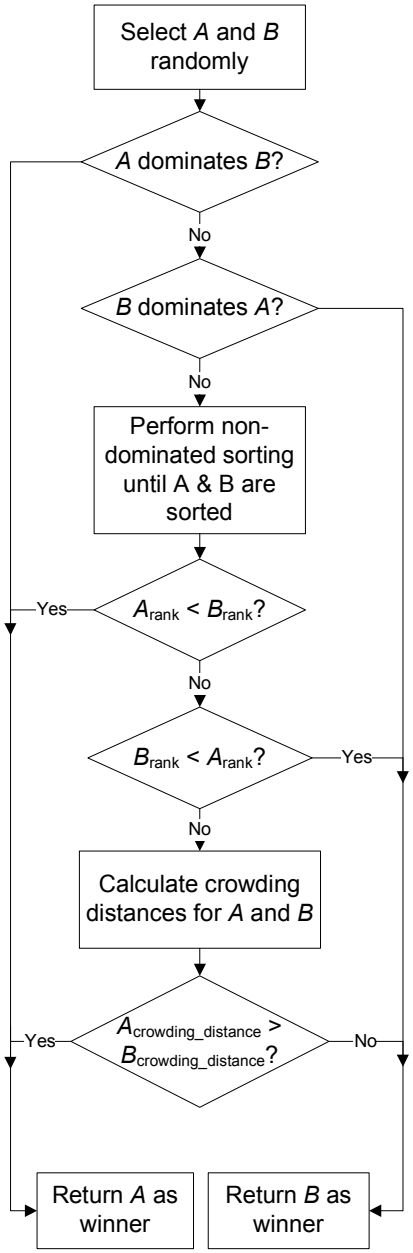
$(\mu + \lambda)$ In this strategy, the selection of individuals to form the next generation of the population takes place among the augmented set of μ parents and λ offspring.

(μ, λ) In this strategy, the selection of individuals to form the next generation of the population takes place among offspring only and all parents are discarded.

Genetic Algorithms Genetic algorithms became widely recognised in the early 1970s (Holland, 1975). In a genetic algorithm, μ offspring are generated from μ parents. The parental selection process is fitness-based and individuals with high fitness have a higher probability of breeding the next generation of the population. Different methods exist for the selection of parents. One example is *tournament selection*, in which a "tournament" is performed among a few individuals chosen at random, and the one with the best fitness is selected as winner. In a tournament selection individuals with worse fitness may also be selected, which prevents premature convergence. It is common that the best individual among the parents are carried over to the next generation unaltered, a strategy known as elitism.

Appendix C

Modified crowding tournament selection



Appendix D

Confidence-based dynamic resampling

```
Select two solutions
p1←Select(P)
p2←Select(P)
while true do
  Set minimum confidence level based on best rank of the two solutions
  P.NonDominatingSort()
  bestRank←Max(p1.rank, p2.rank)
  minConfLevel←ConfLevels(bestRank)
  Check if an overlap exists within the given confidence level
  if Overlap(p1, pC2, minConfLevel) AND then
    (p1.numRepl <= p1.maxRepl OR p2.numRepl <= p2.maxRepl )
    Always replicate each solution at least twice
    if p1.numRepl < 2 then
      BeginSimulation(p1)
    else if p2.numRepl < 2 then
      BeginSimulation(p2)
    else
      Check in which objective the largest standard deviation exist
      o←LargestOverlap(p1, p2, minConfLevel)
      The solution with the largest sample standard deviation in the objective with
      largest objective should be replicated in the first place
      if p1.SampleStd(o) >= p.SampleStd(o) AND p1.numRepl <= p1.maxRepl
      then
        BeginSimulation(p1)
      else if p2.numRepl <= p2.maxRepl then
        BeginSimulation(p2)
      else
        BeginEvaluation(p1)
      end if
    end if
  else
    There is no overlap, return the best solution
    return SelectBest(p1, p2)
  end if
end while
```

```
Function to check if an overlap exists between two solutions within the given confidence level
function Overlap(p1, p2, confLevel)
All solutions must be replicated at least twice, otherwise it is not possible to calculate the overlap
if p1.numRepl <= 1 OR p2.numRepl <= 1 then
  return true
end if
i ←0
while i < numObj do
  Use Welch confidence interval to check if the objectives values overlap
  if WelchConfidenceIntervalOverlap(p1.obj[i], p2.obj[i], confLevel) then
    return true
  end if
  i ←i+1
end while
return false
```

```
Function to check in which objective the largest standard deviation exist
function LargestOverlap(p1, p2, confLevel)
Calculate the confidence interval for the first objective
minMaxInterval←WelchConfidenceInterval(p1.obj[0], p2.obj[0], confLevel)
Set the largest overlap to be in the first objective
largestOverlap←Min(Abs(minMaxInterval[0]), Abs(minMaxInterval [1]))
obj←0
Check if there is a larger overlap in any other objective
i ←1
while i < numObj do
  minMaxInterval←WelchConfidenceInterval(p1.obj[i], p2.obj[i], confLevel)
  if Min(Abs(minMaxInterval[0]), Abs(minMaxInterval [1])) > largestOverlap then
    obj←i
  end if
  i ←i+1
end while
return obj
```

Appendix E

Photos from Volvo Aero

Figure E.1: Overview of production cell at Volvo Aero.

Figure E.2: Operator in the production cell at Volvo Aero.

Figure E.3: Operator loading workpiece into the production cell at Volvo Aero.

Appendix F

Photos from Volvo Cars Engine

Figure F.1: Production line at Volvo Cars Engine.

Figure F.2: Processing machines at Volvo Cars Engine.

Figure F.3: Storage area at Volvo Cars Engine.

Appendix G
More on the Volvo Aero optimisation

In Figure G.1, an example of a Pareto front found by MOPSA-EA in the Volvo Aero optimisation is shown (black points)[1]. As the figure illustrates, there is an obvious conflict between the two objectives of maximising utilisation and minimising tardiness; an improvement in one of the objectives corresponds to a deterioration of the other objective. For comparison, a solution found by domain experts through manual optimisation is also shown in the figure (grey rectangle). It is noticeable that several solutions found by MOPSA-EA dominate this solution.

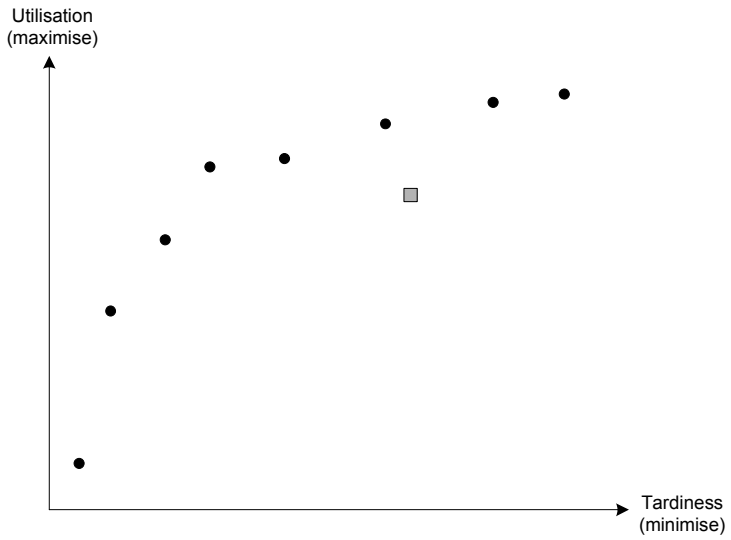


Figure G.1: Example of Pareto front in Volvo Aero optimisation.

[1] With respect to the integrity of the company, the axes of the diagram in the figure have not been numbered.

An interesting aspect of the optimisation results is how far the obtained front is from the "optimal" front. Although the true optimal Pareto front is unknown, it is at least possible to get an idea of how much further the front can move by performing a large number of additional simulations. To investigate this, the optimisation was run until at least 100 succeeding simulations resulted only in small changes of the front, which would indicate that the optimisation has converged. This happened after 2500 simulations, when the optimisation was stopped. Figure G.2 shows the results of this extended optimisation (grey points) in comparison to the original optimisation with 400 simulations (black points). As the figure illustrates, the front obtained in the extended optimisation is quite far from the previous front. This indicates that further improvements of the optimisation results are possible if the optimisation time budget is made more generous.

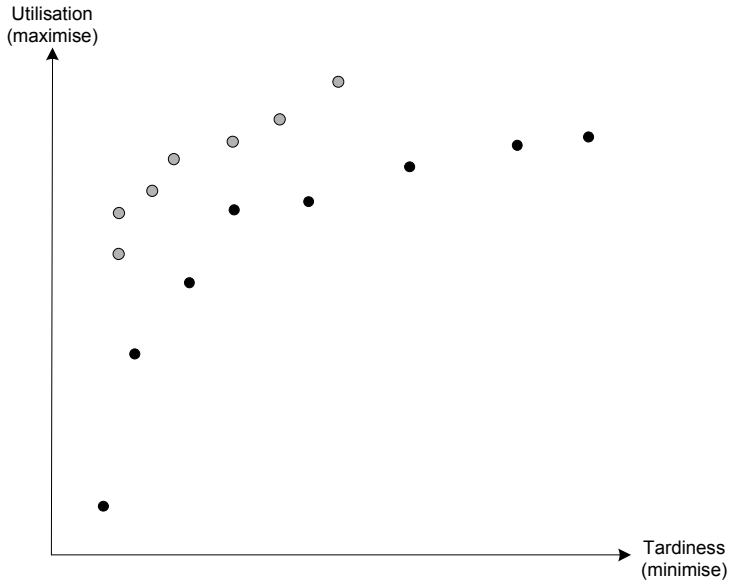


Figure G.2: Extended optimisation of Volvo Aero problem.

Another interesting aspect of the optimisations to study is how much the use of a surrogate (in this case and artificial neural network) actually improves the results. To evaluate the impact of surrogates, MOPSA-EA was applied without the surrogate as a test. The result without the surrogate is shown in Figure G.3 (white points) in comparison to the results with the surrogate (black points). As can be seen in the

figure, although the algorithm is still able to find quite a good front without using the artificial neural network, the results with it are somewhat better. An explanation of the relatively small gain from using the artificial neural network might be that the problem is complex, hence making it hard to obtain an artificial neural network of high precision. Although the precision of the artificial neural network might not be very high, it can, however, still aid the evolutionary algorithm in filtering out really poor solutions and thereby preventing efforts being wasted on inferior solutions.

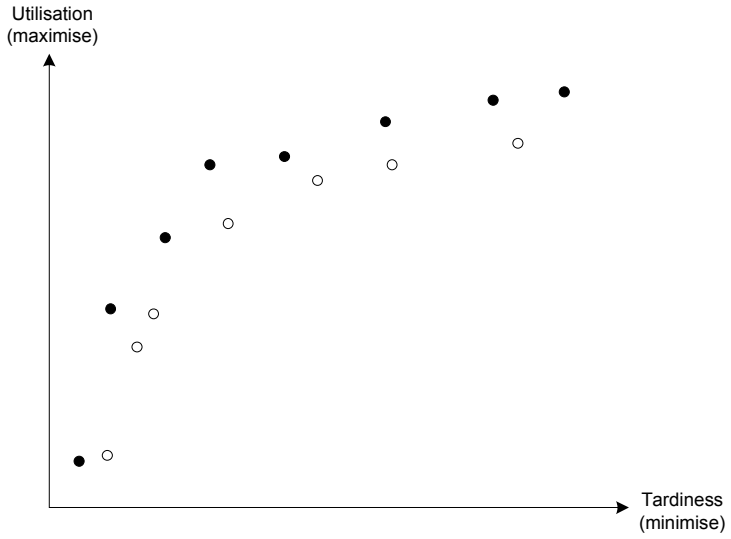


Figure G.3: Surrogate impact in Volvo Aero optimisation.

The artificial neural network is continuously trained during the optimisation and to obtain an indication of its precision over time, mean squared error is measured every time the artificial neural network is re-trained (i.e. every 10th simulation). Mean squared error is the summed square of the difference between the artificial neural network output value and the simulation output value for all samples (cf. Equation 2.4). The mean squared error of the artificial neural network in the Volvo Aero problem is shown in Figure G.4 (screenshot from the Optimise platform). As the figure indicates, the mean squared error value is high in the beginning of the optimisation and quickly decreases when the training set is filled with more solutions. When the training set is full after about 50 simulations, the mean squared error value starts to become stable and then stays relatively constant during the optimisation. The stable mean squared

error value might be explained by two factors: (a) the size of the training data set is constant, and (b) the training data is normalised, which means that although the actual error of the artificial neural network decreases as the optimisation converges towards a narrow area of the search space (assuming that it is easier for an artificial neural network with a limited number of training samples to learn a smaller part of the fitness landscape), the mean squared error stays unchanged.

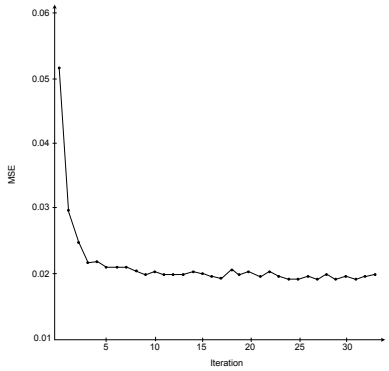


Figure G.4: Mean squared error of artificial neural network over time in Volvo Aero problem.

Appendix H
More on the Volvo Cars Engine optimisation

An example of a Pareto front found by MOPSA-EA in the Volvo Cars Engine optimisation is shown in Figure H.1[2]. As in the Volvo Aero problem, the conflict between the two optimisation objectives of minimising shortage and maximising throughput is obvious. Since there are too many input parameters of this problem (about 500) to perform a manual optimisation, no solution has been provided by the domain experts to compare with.

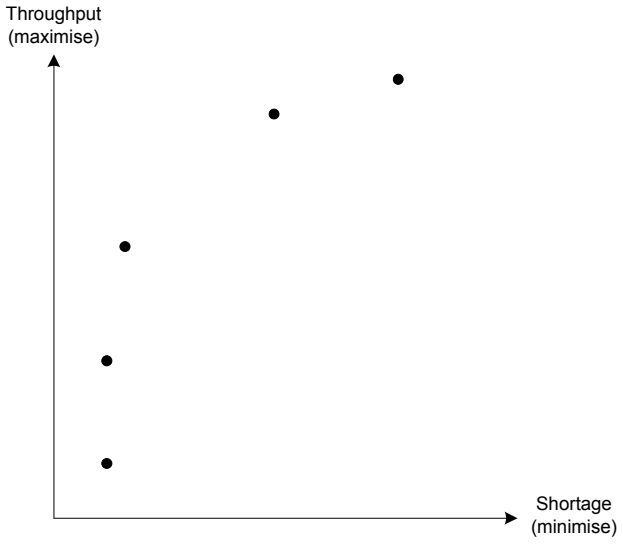


Figure H.1: Example of Pareto front in Volvo Cars Engine optimisation.

[2] With respect to the integrity of the company, the axes of the diagram in the figure have not been numbered.

To get an idea how far the obtained front is from the "optimal" one, the optimisation was run for 5300 simulations, of which the last 100 simulations resulted only in small changes of the front. This would indicate that the optimisation has converged after this number of simulations. In Figure H.2, the results of the extended optimisation (white points) is shown in comparison to the original optimisation with 600 simulations (black points). The difference in the results is even more obvious than in the Volvo Aero problem; quite large improvements in the optimisation results can be achieved if the optimisation time budget is extended and more simulations are allowed.

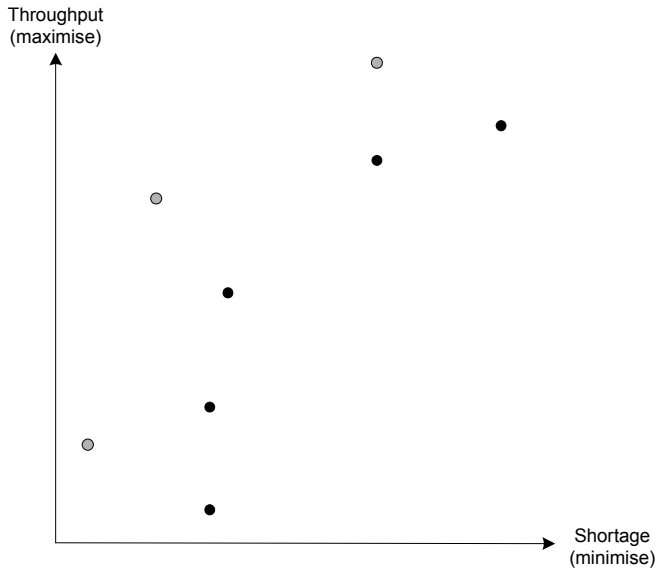


Figure H.2: Extended optimisation of Volvo Cars Engine problem.

To evaluate the impact of the surrogate in the Volvo Cars Engine problem (in this case a C# surrogate model), an optimisation using MOPSA-EA was performed without the surrogate. In Figure H.3, the result without the surrogate (white points) is shown in comparison to the results with the surrogate (black points). If comparing Figure G.3 and Figure H.3, it can be seen that there is a larger benefit of using the surrogate in the Volvo Cars Engine problem compared to the Volvo Aero problem. This indicates that the surrogate in the Volvo Cars Engine problem has a better precision than the surrogate in the Volvo Aero problem.

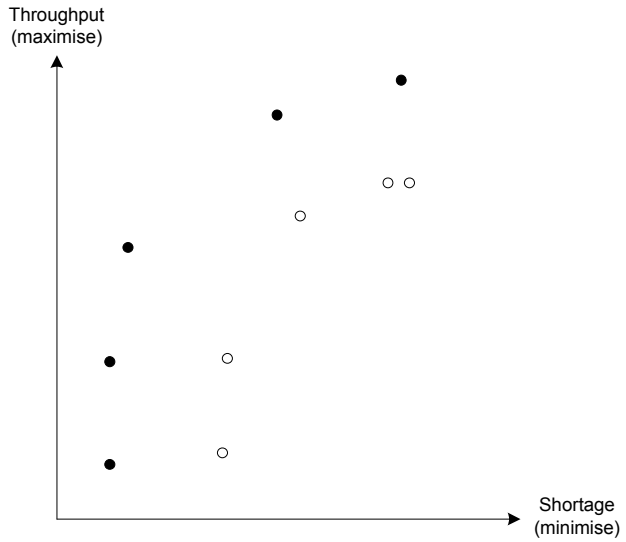


Figure H.3: Surrogate impact in Volvo Cars Engine optimisation.

Appendix I
List of publications

The continuous development of this thesis is reported in the publications listed below (chronologically ordered). All papers with Syberfeldt/Persson[3] as the first author are available for download at www.his.se/pesn. The five most important papers are also included at the end of the appendix.

Journal papers

- **Syberfeldt, A.**, Ng, A., John, R.I. and Moore, P. (2009) Multi-Objective Evolutionary Optimisation of a Real-World Manufacturing Problem. Accepted for publication in *Journal of Robotics and Computer Integrated Manufacturing*.

- **Syberfeldt, A.**, Grimm, H., Ng, A., John, R.I. and Moore, P. (submitted) Evolutionary Optimisation of Noisy Multi-Objective Problems Using Confidence-Based Dynamic Resampling. Submitted to European Journal of Operational Research.

Book chapters

- **Syberfeldt, A.**, Grimm, H., Ng, A., John, R.I. and Moore, P. (submitted) Multi-Objective Evolutionary Optimisation of Computationally Expensive Problems. Submitted to *Computational Intelligence in Expensive Optimization Problems*.

- **Persson, A.**, Grimm, H. and Ng, A. (2008) A Metamodel-Assisted Steady-State Evolution Strategy for Simulation-Based Optimization. *Current Trends in In-*

[3] The author changed her name from Persson to Syberfeldt in fall 2007.

telligent Systems and Computer Engineering, Series: Lecture Notes in Electrical Engineering (vol. 6). O. Castillo, X. Li and A. Sio-Iong (Eds.). Springer. ISBN: 978-0-387-74934-1, pp. 1-14.

- Ng, A., Grimm, H., Lezama, T., **Persson, A.**, Andersson, M. and Jägstam, M. (2008) Web Services for Metamodel-Assisted Parallel Simulation Optimization. *Advances in Communication Systems and Electrical Engineering*, Series: Lecture Notes in Electrical Engineering (vol. 4). Huang, Y-S. Chen and S-I. Ao Springer. X. (Eds.). Springer. ISBN: 978-0-387-74937-2, pp. 281-296.

Conference papers

- **Syberfeldt**, A., Grimm, H., Ng, A., Andersson, M. and Karlsson, I. (2008) Simulation-Based Optimization of a Complex Mail Transportation Network. In *Proceedings of the 2008 Winter Simulation Conference (WSC'08)*, Miami, Florida, USA, 7-10 December, 2008.

- **Syberfeldt**, A., Grimm, H. and Ng, A. (2008) Multi-Objective Simulation-Based Optimisation of Production Systems with Consideration to Noise. In *Proceedings of Swedish Production Symposium 2008 (SPS'08)*, Gothenburg, Sweden, November 18-20, 2008.

- **Syberfeldt, A.**, Grimm, H., Ng, A. and John, R.I. (2008) A Parallel Surrogate-Assisted Multi-Objective Evolutionary Algorithm for Computationally Expensive Optimization Problems. In *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC'08)*, Hong Kong, June 1-6, 2008, pp. 3177-3184.

- **Syberfeldt, A.**, Grimm, H. and Ng, A. (2008) Design of Experiments for Training Metamodels in Simulation-Based Optimisation of Manufacturing Systems. In *Proceedings of The 18th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM'08)*, Skövde, Sweden, June 30-July 2, 2008, pp. 1114-1121.

- Ng, A., **Syberfeldt, A.**, Grimm, H. and Svensson, J. (2008) Multi-Objective Simulation Optimization and Significant Dominance for Comparing Production Control Mechanisms. In *Proceedings of The 18th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM'08)*, Skövde, Sweden, June 30-July 2, 2008, pp. 1210-1219.

- Andersson, M., **Persson, A.**, Grimm, H. and Ng, A. (2007) A Web-Based Simulation Optimization System for Industrial Scheduling. In *Proceedings of the 2007 Winter Simulation Conference (WSC'07)*, Washington DC, USA, 9-12 December 2007, 1844-1852.

- **Persson, A.**, Grimm, H., Ng, A. and Andersson, M. (2007) Metamodel-Assisted Simulation-Based Optimisation of Manufacturing Systems. In *Proceedings of 5th International Conference on Manufacturing Research (ICMR 2007)*, Leicester, UK, September 11-13, 2007, pp. 174-178.

- **Persson, A.**, Grimm, H., Ng, A. and Jägstam, M. (2007) A Case Study of Using Simulation and Soft Computing Techniques for Optimisation of Manufacturing Systems. In *Proceedings of Swedish Production Symposium 2007 (SPS'07)*, Gothenburg, Sweden, August 28-30, 2007.

- **Persson, A.**, Grimm, H., Andersson, M. and Ng, A. (2007) Metamodel-Assisted Simulation-Based Optimization of a Real-World Manufacturing Problem. In *Proceedings of The 17th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM'07)*, Philadelphia, USA, June 18-20, 2007, pp 950-956.

- Andersson, M., **Persson, A.**, Grimm, H. and Ng, A. (2007) Simulation-Based Scheduling Using a Genetic Algorithm with Consideration to Robustness: a Real-World Case Study. In *Proceedings of The 17th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM'07)*, Philadelphia, USA, June

18-20, 2007, pp 957-964.

- **Persson, A.**, Grimm, H. and Ng, A. (2007) A Surrogate-Assisted Steady-State Evolution Strategy with a New Offspring Selection Procedure. In *Proceedings of SAIS 2007, The 24rd Annual Workshop of the Swedish Artificial Intelligence Society*. Borås, Sweden, May 22-23, 2007, pp. 185-189.

- **Persson, A.**, Grimm, H. and Ng, A. (2006) On-line Instrumentation for Simulation-based Optimization, In *Proceedings of the 2006 Winter Simulation Conference (WSC'06)*, Monterey, CA, USA, 3-6 December 2006, pages 304-311.

- **Persson, A.**, Grimm, H. and Ng, A., Karlsson, T., Ekberg, J., Falk, S. and Stablum, P. (2006) Simulation-Based Multi-Objective Optimization of a Real-World Operation Scheduling Problem, In *Proceedings of the 2006 Winter Simulation Conference (WSC'06)*, Monterey, CA, USA, 3-6 December 2006, pages 1757-1764.

- **Persson, A.**, Grimm, H. and Ng, A. (2006) Simulation-Based Optimisation Using Global Search and Neural Network Metamodels, In *Proceedings of The 20th annual European Simulation and Modelling Conference (ESM'06)*, Toulouse, France, 23-25 October 2006, pages 182-186.

- **Persson, A.**, Grimm, H. and Ng, A. (2006) Simulation-Based Optimisation Using Local Search and Neural Network Metamodels, In *Proceedings of The 10th IASTED International Conference on Artificial Intelligence and Soft Computing*, Mallorca, Spain, 28-30 August 2006, pages 178-183.

Conference short papers

- **Persson, A.**, Grimm, H., Ng, A. and Jägstam, M. (2007) Using Soft Computing Techniques in the Simulation-Based Optimisation of Postens Mail Distribution. In *Proceedings of SAIS 2007, The 24rd Annual Workshop of the Swedish Artificial Intelligence Society*. Borås, Sweden, May 22-23, 2007, pp. 193-197.

- Ng, A., Grimm, H., **Persson, A.**, Andersson, M. and Jägstam, M. (2007) A Platform for Metamodel-Assisted Parallel Simulation Optimisation using Soft Computing Techniques. In *Proceedings of SAIS 2007, The 24rd Annual Workshop of the Swedish Artificial Intelligence Society.* Borås, Sweden, May 22-23, 2007, pp. 181-184.

- **Persson, A**. and Ng. A. (2006) OPTIMisation using Intelligent Simulation Tools, In *Proceedings of SAIS 2006, The 23rd Annual Workshop of the Swedish Artificial Intelligence Society.* Umeå, Sweden, May 10-12 2006, pages 83-86.

Posters

- **Persson, A.**, Grimm, H. Ng, A. and Jägstam, M. (2006) Simulation-Based Optimization of a Complex Mail Transportation Network (poster), Presented at *The 2006 Winter Simulation Conference (WSC'06)*, Monterey, CA, USA, 3-6 December 2006.

In the following, the five most important papers among those accepted at the time of submission of the thesis are included. The first paper is "Simulation-based optimisation using global search and neural network metamodels", published at the European Simulation and Modelling Conference in 2006. This paper describes the early ideas of combining evolutionary algorithms and surrogates in simulation-based optimisation. The first, and very simple, version of MOPSA-EA is outlined in the paper. The strategy of generating a pool of candidate offspring and use the surrogate to identify the most promising solution are described as part of the algorithm. In the paper, the algorithm is evaluated on a theoretical single-objective problem (the Rosenbrock function).

The second paper is "A metamodel-assisted steady-state evolution strategy for simulation-based optimization", presented at the International Conference on Artificial Intelligence and Applications in 2007 and published by Springer in 2008. In

this paper, the problem of surrogate imprecision is discussed and initial ideas how to consider it are presented. The technique for considering surrogate imprecision described in the paper is quite complicated and probably not very efficient, but still important since it provides the foundation for the current technique used in MOPSA-EA. In the paper, two real-world problems are used in the evaluation. Both problems have two objectives to be optimised, and these are considered through a weighted sum approach.

The third paper is "A parallel surrogate-assisted multi-objective evolutionary algorithm for computationally expensive optimization problems", published at the IEEE Congress on Evolutionary Computation in 2008. In this paper, several important improvements of the algorithm are described. These include a Pareto approach to multi-objective optimisation, support for parallelism, and a refined strategy for considering surrogate imprecision. Furthermore, the name of the algorithm is set in the paper: "Multi-objective parallel surrogate-assisted evolutionary algorithm" (MOPSA-EA).

The fourth paper is "Multi-objective simulation-based optimisation of production systems with consideration to noise", published at the Swedish Production Symposium in 2008 (an extended version of the paper is currently considered for publication in European Journal of Operation Research, but final acceptance had not yet been confirmed at the time for submission of the thesis). This paper discusses the importance of dealing with noise in simulation-based optimisation of real-world problems, and describes the confidence-based dynamic resampling technique used in MOPSA-EA for this purpose.

The fifth paper is "Multi-objective evolutionary optimisation of a real-world manufacturing problem", accepted for publication in the Journal of Robotics and Computer Integrated Manufacturing in 2009. This paper focuses on MOPSA-EA's applicability in real-world problems and provides an extensive analysis of the algorithm.

SIMULATION-BASED OPTIMISATION USING GLOBAL SEARCH AND NEURAL NETWORK METAMODELS

Anna Persson
Henrik Grimm
Amos Ng
Centre for Intelligent Automation
University of Skövde
Box 408, Skövde
Sweden
E-mail: {anna.persson, henrik.grimm, amos.ng}@his.se

KEYWORDS
Optimisation, Simulation, Global Search, Metamodel, Neural Network

ABSTRACT

This paper presents a new population-based metaheuristic algorithm for simulation-based optimisation. The proposed algorithm uses metamodels for efficiency enhancement. Similar to other population-based metaheuristics, such as Genetic Algorithms (GA), it generates and maintains a population of solutions and progresses incrementally generation by generation using genetic operations. The difference is that a trained metamodel is used to discard inferior candidate solutions while keeping the most promising ones. This could significantly enhance the efficiency of the optimisation process by avoiding time-consuming simulation runs for the candidate solutions that lack potential. During the optimisation, the accuracy of the metamodel is constantly improved as on-line training is applied after each generation of solutions have been simulated. The proposed algorithm is implemented on a benchmark optimisation problem and initial results show that the algorithm is able to effectively enhance the performance of the simulation-based optimisation process in comparison with a standard GA-based approach.

1 INTRODUCTION

One of the most important and challenging subjects in the simulation field today is simulation-based optimisation (Buchholz 2005). It has shown to be a powerful technique for systems improvement and has been successfully applied to address a wide range of real-world industrial optimisation problems (April et al. 2004). The general problem in simulation-based optimisation (SO) is to find a setting of decision variables that maximize or minimize a given objective function, assuming that it cannot be computed analytically but must be estimated through simulation.

Population-based optimisation methods such as Genetic Algorithms and Evolutionary Strategies are powerful search algorithms commonly found in SO. These algorithms are increasingly being used to solve a wide range of different optimisation problems, especially when there are a large number of parameters with complex dependencies. However, the main weakness of using population-based optimisation methods in SO is that they require a large number of fitness evaluations. Typically, a population-based optimisation strategy requires thousands of simulation evaluations and it is not uncommon for simulation models to run for hours. For practical applications of SO it is of critical importance that the optimisation process is constrained within reasonable time limits and the efficiency of the optimisation process is crucial.

One potential way to enhance the efficiency of SO and reduce the number of time-consuming simulation runs is to employ computationally cheap metamodels (Alam et al. 2004). Metamodels, also known as surrogate or approximate models, are essentially a "model of the model" which may be used to approximate a simulation model. By adopting metamodels, the computational burden of the optimisation process can be greatly reduced since the computational cost associated with using a metamodel is much lower than the standard approach of performing simulation runs for all configurations generated by the optimizer.

This paper presents a new population-based optimisation algorithm for SO that uses metamodels for efficiency enhancement, called Metamodel Enhanced (MME) Global Search. Although the basic motivation for MME Global Search is to be used in SO, it can be applied to any general optimisation scenario in which a computationally expensive evaluation function can be approximated by some fast metamodel. In the demonstration of the algorithm in this paper an Artificial Neural Network (ANN) based metamodel is used, however any adaptive metamodeling technique is possible. The intended application of the MME Global Search Algorithm is primarily on complex problems that are suitable to be solved with global search techniques, for example, those with multiple optima.

2 RELATED WORK

ANNs are mathematical models that attempt to imitate the behaviour of biological brains. ANNs have universal approximation characteristics and also the ability to adapt to changes through training. Instead of following a set of rules, they are able to learn underlying relationships between inputs and outputs from a collection of training examples and to generalize these relationships to previously unseen data. These attributes make ANN based metamodels very suitable to be used as the substitutes for computationally expensive simulation models.

In most ANN based simulation metamodelling approaches, after the training of the metamodels the

simulation models are completely substitued during the optimisation process. These approaches can only be successful when there is a small discrepancy between the outputs from the metamodel and the simulation. Due to lack of data and the high complexity of real-world problems, it is generally difficult to develop a metamodel with sufficient approximation accuracy that is globally correct and metamodels often suffer from large approximation errors which may introduce false optima (Jin et al. 2002). One way to handle this problem and assure that the optimisation algorithm is not misled when the complexity of the fitness landscape is high is to alternate between the metamodel and the simulation model during the optimisation. Some of the work in combining simulation, ANN metamodels, and population-based optimisation are described in the rest of this section.

Bull (1999) presents an approach where an ANN metamodel is used in conjunction with a costly evaluation function to increase the efficiency of the optimisation. The neural network is first trained with a number of initial samples to approximate a theoretical model and a GA then uses the metamodel for fitness evaluations. For every 50 generations, the fittest individual in the population is evaluated using the original fitness function. This individual replaces the sample representing the lowest fitness in the training data set and the ANN is then retrained. The authors found that the GA is misled by the ANN when the fitness landscape of the modelled system is complex.

Jin et al. (2002) propose an approach for managing metamodels in population-based evolution. The main idea of this approach is that the frequency at which the original function is called and the metamodel is updated are determined by the estimated accuracy of the metamodel. The authors introduce the concept of evolution control and propose two control methods; controlled individuals and controlled generations. With controlled individuals, part of the individuals in a population are chosen and evaluated using the original fitness function. The controlled individuals can be chosen either randomly or according to their fitness values. With controlled generations, the whole population of N generations are evaluated with the original fitness function in M generations ($N \leq M$). On-line learning of the metamodel is applied after each call to the original fitness function when new training data are available. The authors carry out empirical studies to investigate the convergence properties of the implemented evolution strategy using an ANN-based metamodel on two benchmark problems. The authors found that incorrect convergence will occur if the metamodel has false optima.

Khu et al. (2004) discuss the integration between evolutionary algorithms and metamodels and propose a strategic and periodic scheme of updating the metamodel to ensure that the metamodel is constantly relevant as the search progresses. In the suggested approach, the whole population are first evaluated using the metamodel and the best individuals in the population are then evalutated using the true fitness function. The authors implement an ANN metamodel and a genetic algorithm for hydrological model calibration and show that there is a significant advantage in using metamodels for water and environmental system design.

Yan and Minsker (2004) propose a dynamic metamodelling approach, in which ANNs and support vector machines (SVM) are embedded into a GA. Data produced from early generations of the GA are sampled to train the ANN and SVM and the original evalution function is periodically called to dynamically update the ANN and SVM. The authors applied their proposed method to solve groundwater optimisation problems and results from their study show that satisfactory results can be achieved if the ANN metamodel is retrained or updated to fit the GA population in later generations.

Most approaches that make use of global search optimisation, simulations, and ANN-based metamodels employ the metamodel as an evaluation substitute in the ordinary evolutionary process. In contrast to previous studies, this paper presents an approach of using the metamodel for the probing of promising candidates to transfer to the next generation.

3 THE MME GLOBAL SEARCH ALGORITHM

The MME algorithm is a population-based metaheuristic optimisation algorithm. Figure 1 presents the general procedure of the algorithm. The algorithm maintains a population of solutions and progresses in increments called generations, where the population of each generation builds upon the previous generation. The basic principle of the algorithm is to generate a large number of candidate solutions and use a metamodel to choose the most promising ones to transfer to the next generation. There are two assumptions: (1) a good metamodel should be able to dismiss inferior solutions and thus avoid wasting valuable simulation time, and (2) the time required for computing the metamodel is negligible when compared to a simulation run. During the optimisation, the accuracy of the metamodel is constantly improved through applying on-line training after each generation of solutions have been simulated.

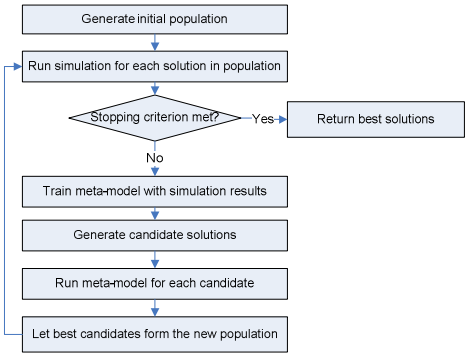


Figure 1: The MME Global Search Algorithm

3.1 Algorithm Core

A solution is defined by a triple $(input, mm\_output, sim\_output)$ where $input$ is an input sample, mm_output is the output produced by the metamodel, and sim_output is the output produced by the simulation. Any of these attributes can be unassigned. For example, if sim_output is unassigned, this means that the solution has not been simulated. In order to refer to the attributes of a solution, a subscript notation is used, e.g., i_{input}

is the *input* attribute of solution i. The algorithm core comprises three functions; MME_Global_Search, Evaluate_Population, and Evaluate_Candidates. The main function is MME_Global_Search, which calls Evaluate_Population for evaluation of solutions in a population and Evaluate_Candidates for evaluating candidate solutions in the population (Figure 2).

```
function MME_Global_Search( )
Returns: Best solutions found.
  g ← 0
  population_0 ← Generate_Initial_Population( )
  Evaluate_Population( population_0 )
  Train_Meta_Model({ population_0 })
  while (not Stop_Optimization( )) do
    g ← g +1
    candidates_g ← Generate_Candidates( population_{g-1} )
    Evaluate_Candidates( candidates_g )
    population_g ← Choose_Solutions( population_{g-1}, candidates_g )
    Evaluate_Population( population_g )
    Train_Meta_Model({ population_0, …, population_g })
  end
  return Best_Solutions( ⋃_{i=0}^{g} population_i )

function Evaluate_Population( population )
  foreach p in population do
    if p_output = null then
      p_output ← Run_Simulation( p_input )
    end
  end

function Evaluate_Candidates( candidates )
  foreach p in candidates do
    p_rough_output ← Run_Meta_Model( p_input )
  end
```

Figure 2: Algorithm Core Functions

3.2 Problem-Specific Functions

The algorithm calls a number of problem-specific functions, which are described in this section.

function Generate_Initial_Population( )

Returns: A set of input samples.

Generates an initial population for the algorithm. One way to do this is to randomly generate a set of points in the search space. Alternatively, a more structured approach, such as Design of Experiments, can be used. In some cases, fairly good solutions already exist (e.g., from earlier optimisations) and can be directly returned from this function.

function Stop_Optimization( )

Returns: True to stop the optimisation.

This can, for example, be based on the quality of the solutions, time passed, or on the number of iterations since the best solution was found.

function Generate_Candidates(*population*)

Input: A population of solutions.
Returns: A set of candidate solutions.

Generates a set of candidate solutions from a population. The returned set of candidate solutions should normally be much larger than the population. The way that new solutions is generated is dependant on the solution input representation and may also include problem-specific heuristics. New solutions may, for example, be generated by using a combination of crossover and mutation operators (as used by a Genetic Algorithm). A specific implementation of this function is described in Section 3.3.

function Choose_Solutions(*previous_population, candidates*)

Input: The previous population and a new set of candidate solutions.
Returns: A set of solutions.

This function chooses the most promising solutions to use as the population for the next generation. This is normally based on the fitness of the solutions, but may also penalize similar solutions to keep the population diversified. The reason that the best candidates in the previous population is passed to the function is to support elitism, i.e. to keep some promising solutions for the new population. Through the conditional check in Evaluate_Population, solutions that are kept from the previous generation have no need to be simulated again.

function Best_Solutions(*solutions*)

Input: A set of solutions.
Returns: The best solutions.

Returns the best solutions based on user-defined criteria, such as the Pareto front.

function Run_Simulation(*input*)

Input: An input sample.
Returns: Output response from simulation.

Runs the accurate, but time-consuming, simulation.

function Train_Meta_Model(*populations*)

Input: A set of simulated populations.

Trains the metamodel with the given solutions or a subset thereof.

function Run_Meta_Model(*input*)

Input: An input sample.
Returns: Output response from metamodel.

This function is assumed to be many orders of magnitude faster than Run_Simulation. Note that the structure of the output returned from this function may not be the same as returned from Run_Simulation. The metamodel may, for example, return an objective value as output instead of the output type returned by the simulation.

3.3 Specialisation of Algorithm

This section describes a specific implementation of the function Generate_Candidates based on the concepts of Genetic Algorithms (Figure 3). The function contains two constants, *num_candidates* and *crossover_frequency*. The constant *num_candidates* specifies the number of candidate solutions to generate and it is assumed to be an even number. Three problem-specific functions are used, which are explained below.

function Select(*population*)

Input: A set of solutions.
Returns: One of the solutions from the set.

Selects a solutions based on its fitness in relation to the other solutions. A number of popular selection schemes exists, such as roulette wheel, ranking, and tournament.

function Crossover(*solution1*, *solution2*)

Input: Two parent solutions.
Returns: Two child solutions.
Generates two new solutions based on a recombination of the two parent solutions.

function Mutate(*solution*)

Input: A solution.
Returns: A mutated solution.
Returns a solution that is a mutated version of the given solution.

function Generate_Candidates(*population*)
Returns: `Returns` *num_candidates* `candidates.`
  candidates ← ∅
  for *i* ← 1 **to** $\frac{num\_candidates}{2}$ **do**
    a ← Select(*population*)
    b ← Select(*population*)
    with probability *crossover _ frequency* **do**
      $(c,d) \leftarrow \text{Crossover}(a,b)$
    else
      $(c,d) \leftarrow (a,b)$
    end
    $candidates \leftarrow candidates \cup \{\text{Mutate}(c), \text{Mutate}(d)\}$
  end
  return *candidates*

Figure 3: Variant of Algorithm

4 BENCHMARK TEST PROBLEM

This section demonstrate the MME Global Search algorithm applied to the 2-D Rosenbrock optimisation benchmark problem (Equation 1). This is used as an initial benchmark function by assuming that it represents a computationally expensive evalution function.

$$f(x,y) = 100\left(y - x^2\right)^2 + (1-x)^2 \qquad (1)$$

The fitness landscape of the Rosenbrock function (plotted in Figure 4) has a global minimum of 0 at the point (1, 1).

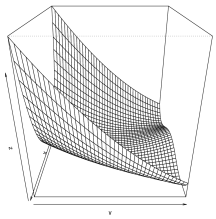


Figure 4: The 2-D Rosenbrock Function

An ANN is developed as a fast metamodel of the Rosenbrock function. It is trained to estimate the Rosenbrock function as a function of a search space coordinate. A feed-forward network with two hidden layers is constructed with two input nodes, 20 nodes in each hidden layer and one output node. While a single hidden layer may be sufficient to approximate any continuous function, it is not always optimal in terms of learning time (Chester 1990). One hidden layer may require an infinite number of neurons to approximate a given function and the use of two hidden layers can avoid this assumption. The data set which is used to train the ANN consists of 1000 input-output pairs randomly generated in the search space.

4.1 Implementation

An input sample consists of a vector of 2 real values, corresponding to a X-Y coordinate in the search space. An output sample consists of a real value, corresponding to the Rosenbrock function value. The goal of the optimisation is to minimise this value. The implementation uses the GA-based variant of Generate_Candidates described in Section 4.3. A population size of 10 indivudals is used and the initial population is randomly initated. In each iteration of the algorithm, 100 candidate solutions are generated and evaluated using the ANN. Solutions are selected for reproduction using tournament selection in which two solutions are randomly chosen and the one with the lowest objective function value is returned. A standard one-point crossover is used and recombination of solutions occurs with a probability of 0.5. Each value in the solutions is mutated using a Gaussian distribuation with a deviation that is randomly selected from the interval [0,10] for each individual. During the search, the ANN is trained continuously with incremental training using back-propagation for 1 epochs with a learning rate of 0.1. Since the training set in this test is large, only data from the last evaluated population is used to train the network and the training continues from the last weights. The optimisation runs for 100 iterations before it terminates and returns the best solution found.

4.2 Results

This section presents the results of the MME Global Search implementation described in the previous section. For comparison, a standard GA is implemented for the same optimisation problem and simulation model. This algorithm uses the same representation, objective function, crossover operator and mutation operator as the MME Global Search implementation.

In Figure 5, average results from 5000 replications of the two experiments are shown. The chart shows the best fitness value found against the number of function evaluations.

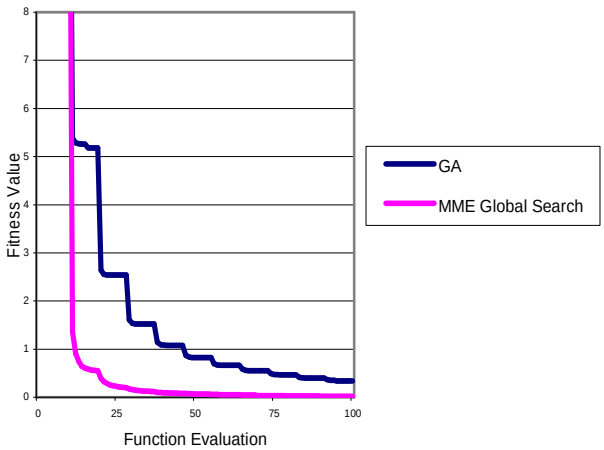


Figure 5: Comparison of Experiments

As the chart shows the MME Global Search algorithm converge faster than the standard GA.

Using on-line training, the accuracy of the metamodel is continuously improved, as shown in Figure 6. This figure presents the Mean Square Error (MSE) of the metamodel estimated locally based on the information from the last simulation. The MSE presented is an average of 5000 replications.

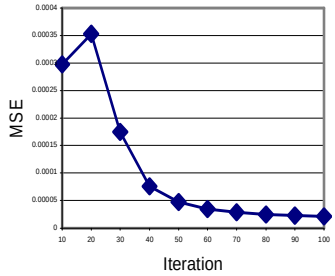


Figure 6: Estimated Metamodel MSE

5. CONCLUSIONS AND FUTURE WORK

This paper presents a new population-based meta-heuristic algorithm for simulation-based optimisation. The proposed algorithm uses a metamodel for efficiency enhancement. Similar to other population-based metaheuristics, such as Genetic Algorithms (GA), it generates and maintains a large population of solutions and progresses incrementally generation by generation using genetic operations. The difference is that it generates a large set of candidate solutions followed by the use of a metamodel to probe for the most promising ones. During the optimisation, the accuracy of the metamodel is constantly improved as on-line training is applied after every individual in a generation of solutions has been simulated.

The proposed algorithm is implemented on a benchmark optimisation problem and results show that the efficiency of the optimisation process could be significantly enhanced by avoiding running time-consuming simulations for low-quality candidate solutions. Initial results also indicate that the proposed algorithm shows good performance in comparison with a standard GA-based approach.

Future work will focus on many different aspects of this ongoing research. These include verification of the proposed algorithm by applying it to different real-world optimisation problems with various properties, and performing further benchmarkings with some other state-of-the-art algorithms. Planned future work also includes combining the algorithm with the metamodel-based local search algorithm presented in Persson et al. (2006).

An intresting variant of the algorithm is to choose the *n* best candidate solutions to transfer to the next generation not exclusively based on fitness values. When there is a large number of candidate solutions, there is a risk that a great proportion of the generated candidates are identical to one another and as a consequence the new population becomes more or less homogeneous. To prevent this, a diversification control can be implemented, punishing candidate solutions that are too close to each other. Planned future work will include the testing of this variant of the proposed algorithm.

When using ANN and population-based search methods a large number of variables exist, such as number of hidden layers, number of nodes in the hidden layers, population size, mutation rate, etc. In future work, the effects of varying these parameters will be tested. Further, for ANNs the data samples in the intial training set can have large influence on the networks' approximation ability. Ways to achieve more effective training of the ANN will be investigated in the future.

The higher the accuracy of the metamodel, the more frequently it can be used and hence the time consumption of the optimisation process can be reduced. However, it is very difficult to estimate the global accuracy of the metamodel and future work includes investigating how one can get an understanding of the metamodels influence on the search direction. Currently the whole population of solutions is evaluated using the simulation model, but with an adequate metamodel it may, for example, be sufficient to evaluate only a subset of the population with the simulation and using the metamodel for evaluating the other part.

REFERENCES

Alam, F., K.R. McNaught; and T.J. Ringrose. 2004. "A comparison of experimental designs in the development of a neural network simulation metamodel". *Simulation Modelling Practice and Theory*, Vol.12 No.7-8, 559–578.

April, J, M. Better, F. Glover; and J. Kelly. 2004. "New advances for marrying simulation and optimization". In *Proceedings of the 2004 Winter Simulation Conference* (Washington, D.C., Dec.5-8), 80-86.

Buchholz, P. and A. Thümmler. 2005. "Enhancing evolutionary algorithms with statistical selection procedures for simulation optimization". In *Proceedings of the 2005 Winter Simulation Conference* (Orlando, FL, Dec.4-7), 842-852.

Bull, L. 1999. "On model-based evolutionary computation". *Software Computing*, Vol.3, 76-82.

Chester, D. 1990. "Why two hidden layers are better than one". In *Proceedings of the International Joint Conference on ANNs* (San Diego, June 17-21), 265-268.

Jin, Y., M. Olhofer; and B. Sendhoof. 2002. "A Framework for Evolutionary Optimization With Approximate Fitness Functions". *IEEE Transactions on Evolutionary Computation*, Vol.6 No.5 (Oct), 481-494.

Khu, S.T., D. Savic, Y. Liu; and H. Madsen. 2004. "A fast Evolutionary-based Metamodelling Approach for the Calibration of a Rainfall-Runoff Model". In *Proceedings of the First Biennial Meeting of the International Environmental Modelling and Software Society* (Osnabruck, Germany), 147-152.

Persson, A., Grimm, H. and Ng, A. 2006. "Simulation-Based Optimisation Using Local Search and Neural Network Metamodels". *In Proceedings of the 10th IASTED International Conference on Artificial Intelligence and Soft Computing* (Mallorca, Spain), 178-183.

Yan, S. and B. Minsker. 2004. "A Dynamic Metamodel Approach to Genetic Algorithm Solution of a Risk-Based Groundwater Remediation Design Model". In *Proceedings of World Water and Environmental Resources Congress* (Salt Lake City, Utah, June 27-July 1), 1-10.

A Parallel Surrogate-Assisted Multi-Objective Evolutionary Algorithm for Computationally Expensive Optimization Problems

Anna Syberfeldt, Henrik Grimm, Amos Ng, and Robert I. John

***Abstract*— This paper presents a new efficient multi-objective evolutionary algorithm for solving computationally-intensive optimization problems. To support a high degree of parallelism, the algorithm is based on a steady-state design. For improved efficiency the algorithm utilizes a surrogate to identify promising candidate solutions and filter out poor ones. To handle the uncertainties associated with the approximative surrogate evaluations, a new method for multi-objective optimization is described which is generally applicable to all surrogate techniques. In this method, basically, surrogate objective values assigned to offspring are adjusted to consider the error of the surrogate. The algorithm is evaluated on the ZDT benchmark functions and on a real-world problem of manufacturing optimization. In assessing the performance of the algorithm, a new performance metric is suggested that combines convergence and diversity into one single measure. Results from both the benchmark experiments and the real-world test case indicate the potential of the proposed algorithm.**

I. Introduction

EVOLUTIONARY algorithms (EAs) are powerful techniques for solving complex optimization problems [1]. While traditional analytical optimization methods have been unable to cope with the challenges imposed by many real-world systems, such as multimodality, non-separability and high dimensionality, EAs have proven to be highly useful in searching reasonably good solutions. During the years, EAs have shown to produce convincing results for a wide variety of problems such as engineering design, operational planning, and scheduling.

When applying EAs in real-world scenarios, the simultaneous optimization of more than one objective is often required. The difficulty with problems of multiple objectives is that there is usually no single optimal solution with respect to all objectives, as improving the performance on one objective would deteriorate the performance of one or more of the other objectives. Instead of a single optimum, there is a set of optimal trade-offs, known as the Pareto set. For deriving the Pareto set, most multi-objective optimization algorithms use the concept of dominance [1]. A solution *x*' is said to dominate another solution *x*'' if *x*' is no worse than *x*'' in all objectives, and *x*' is strictly better than *x*'' in at least one objective. The solutions that are not dominated by any other solutions are called Pareto-optimal and form the Pareto set. Solutions can also be sorted into different non-dominated ranks; rank 1 is made up of the Pareto-optimal solutions among the complete set of solutions, rank 2 of the Pareto-optimal solutions identified when temporarily discarding all solutions associated with rank 1, etc. Since EAs maintain a population of solutions, it is possible to capture multiple Pareto-optimal solutions in one single optimization run. This makes EAs very suitable for the handling of multi-objective problems [1].

Although EAs have achieved great success in many applications, these algorithms have also encountered some technical hurdles. Among these, efficiency is a major challenge. Real-world optimization problems often involve an immense number of possible solutions, and an EA needs a large number of simulation evaluations before an acceptable solution can be found [2]-[3]. Even with improvements in computer processing speed, one single simulation evaluation may take a couple of minutes to hours or days of computing time [4]-[5]. This poses a serious hindrance to the practical application of EAs in real-world scenarios, and to tackle this problem various approaches have been suggested. A commonly used technique to address the problem of computationally expensive simulations is parallelization. With parallel processing nodes, several solutions can be effectively simulated simultaneously. Another technique for increased efficiency is the incorporation of computationally efficient surrogates (also called metamodels). A surrogate is an approximation of the simulation; if the simulation is represented as $y = f(x)$, then a surrogate is represented as $\hat{y} = \hat{f}(x)$, such that $\hat{f}(x) = f(x) + e(x)$, where $e(x)$ is the approximation error. For constructing surrogates, a variety of different techniques have been proposed, among the most popular in evolutionary optimization being Artificial Neural Networks, Radial Basis Function Networks, and Kriging models [6]. The application of surrogates to EAs is, however, not completely straightforward. Constructing a surrogate that represents the complete search space correctly is hard, especially in real-world problems with sparse data samples [6]. If not handled properly, the error of the surrogate may mislead the EA to propagate inferior individuals; weak offspring might be chosen for the next generation while good ones might be excluded. For successful results, the imprecision of the surrogate must

Manuscript received March 1, 2008. This work was supported in part by the Knowledge Foundation, Sweden and Volvo Aero, Sweden.

A. Syberfeldt, H. Grimm, and A. Ng are with the Centre for Intelligent Automation, University of Skövde, Skövde, PO 54148 Sweden (corresponding author to provide phone: 46-500-448577; fax: 46-500-448598; e-mail: {anna.syberfeldt, henrik.grimm, amos.ng}@his.se).

R. I. John is with the Centre for Computational Intelligence, De Montfort University, Leicester LE1 9BH, UK (e-mail: rij@dmu.ac.uk).

therefore be considered in the optimization; otherwise it is very likely that the search would converge to a false optimum [3].

In this paper, a new multi-objective EA that incorporates a surrogate for solving computationally-intensive optimization problems is described. This algorithm is based on a steady-state design to support a high degree of parallelism. Most multi-objective EAs use a generational approach [5], which is not optimal with respect to parallel efficiency. In generational algorithms, results for a complete generation must be awaited in order for the search to proceed. This is inefficient if the population size is not divisible by the number of processing nodes, or if simulations on different nodes take different amounts of time. Furthermore, if the population size is less than the number of processing nodes, all computing resources will not be utilized. In comparison, a steady-state design enables a higher degree of parallelism, since new solutions are continuously created and the number of parallel evaluations is not limited by the population size. Besides their parallel efficiency, steady-state algorithms have also been considered being more efficient on complex optimizations problems and are able to find good solutions in less time compared to generational algorithms [7].

In the proposed algorithm, the surrogate is used to screen candidate solutions and identify the most promising ones. Instead of generating only a single offspring, which is normally done in steady-state algorithms, a pool of multiple offspring is created. Each of the offspring is evaluated by the surrogate, and the best one is simulated and inserted into the population. In the selection of the offspring to include in the population, the imprecision associated with the surrogate is considered using a new approach for multi-objective optimization.

In the next section, the fundamental design of the proposed Multi-Objective Parallel Surrogate-Assisted EA, called MOPSA-EA, is presented.

II. Algorithm Description

The basic algorithm is described with pseudo code in Fig. 1. Initially, the first generation of the population P is filled with μ random solutions. While the population is not full and there are processing nodes available, new random solutions are being created and sent to the simulation for evaluation. When μ solutions have been simulated, offspring generation is initiated. Offspring are created from parents in P chosen using crowding tournament selection (described in [1]). With tournament selection, solutions with worse fitness may also be selected, which maintains diversity in the population and prevents premature convergence. For the tournament, two solutions A and B are chosen randomly and A is declared as the winner over B if either

(i) A has a better rank than B, or

(ii) A and B have the same rank, but A has a larger crowding distance than B.

In the algorithm, a slightly optimized implementation of the standard crowding tournament selection operator is used in which a pre-control is made if A dominates B. By first verifying if a dominance relationship exists between the two solutions, an unnecessary non-dominated sort can be avoided.

When generating offspring, instead of creating only a single new solution from a pair of parents, which is the commonly used strategy in steady-state algorithms, a pool of λ candidate offspring is created. Such an offspring pool, called O, is created as soon as a processing node becomes available. The solutions in O are evaluated by the surrogate, and since the computational cost of surrogate evaluations can be neglected in real-world optimizations [8], the size of the pool might be large. The surrogate objective values assigned to solutions in O are adjusted to take the imprecision of the surrogate into consideration. This is done by modifying the values based on the calculated error of the surrogate (the details of this procedure are described in the next section). Based on the adjusted surrogate objective values, the most promising solution in O to insert into P is selected. In this procedure, all solutions of rank 1 in O are identified (called O_{R1}) and checked for domination against all solutions of rank 1 in P (called P_{R1}). By only identifying O_{R1} and P_{R1}, a full non-dominating sort is avoided. The solution in O_{R1} dominating most solutions in P_{R1} is selected and simulated. If several solutions in O_{R1} share the position of dominating most solutions in P_{R1}, the one having the largest Euclidean distance to its closest neighbor in P_{R1} is selected (note that crowding distance cannot be used since the solutions of O_{R1} and P_{R1} might be of different ranks if merged). Before the selected offspring is inserted into P, the worst solution in P is removed by performing a non-dominated sort and discarding the solution with the smallest crowding distances in the last rank. An elitistic approach is also possible, in which an offspring is only inserted into P if it is not dominated by all solutions in P.

The simulation sample obtained from a newly inserted offspring may be used to update the surrogate. To save time, an update does not need to take place every time a new sample becomes available, but only every *N*:th sample. Since the algorithm is neutral with respect to surrogate modeling technique, it does not specify *how* to update the surrogate. Surrogate update strategies vary between different techniques, and are also highly problem-dependent.

```
function Main( )
  P ← ∅
  iter ← 0
  while (not StopOptimization( )) do
    while (ProcessingNodeAvailable( )) do
      if (|P| = μ) then
        BeginSimulation(GenerateNewSolution(P))
      else
        BeginSimulation(GenerateRandomSolution( ))
      endif
    endwhile
    p ← WaitForFinishedSimulation( )
    if (Mod(iter, surrogateUpdateFrequencey) = 0) then
      surrogate.Update( )
    endif
    if (|P| = μ) then
      P.Remove(Select Worst(P))
    endif
    P.Add(p)
    iter ← iter + 1
  endwhile

function GenerateNewSolution(P)
  O ← ∅
  repeat λ times
    parent1 ← SelectForReproduction(P)
    parent2 ← SelectForReproduction(P)
    o ← Crossover(parent1, parent2)
    Mutate(o)
    SurrogateEvaluation(o)
    O.Add(o)
  end
  return Select Best(O)
```

Fig: 1. Multi-Objective Parallel Surrogate-Assisted Evolutionary Algorithm (MOPSA-EA).

A. Offspring Selection with Consideration to Surrogate Imprecision

A new method for handling surrogate imprecision is used in the offspring selection procedure in MOPSA-EA. Basically, surrogate objective values assigned to offspring are adjusted to consider the error of the surrogate. For each offspring, the objective errors of its parents are calculated by evaluating the parents using the surrogate and taking the difference between their assigned simulation objective values and the obtained surrogate objective values. For example, if the simulation values assigned to a parent are $(51,63)$ and a surrogate evaluation returns the values $(45,77)$, then the error of that parent is $(51,63)-(45,77)=(6,-14)$. Since the accuracy of the surrogate might change dynamically, surrogate values are calculated every time a solution is chosen as parent.

The surrogate objective values of an offspring are modified by adding the weighted mean of the error values of the offspring's parents. The weighting is based on each parent's influence on the child during crossover (Fig. 2). In case no crossover is applied when creating the child (i.e. only mutation is performed), the influence of one of the parents is 100%.

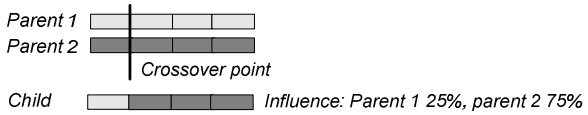


Fig: 2. Example of parents' influence on child.

The adjustment of surrogate objective values can be illustrated by an example: if an offspring is assigned the values $(50,81)$ from the surrogate and the error values of its parents are $(6,-14)$ and $(8,2)$, respectively, then the new objective values of the offspring become $50+0.25*6+0.75*8=56.5$ and $81+0.75*-14+0.25*2=69$ (assuming that parents' influence is 25% and 75%, respectively).

This approach of obtaining offspring error values has similarities with the concept of fitness inheritance, in which an offspring's objective values are based on its parents' objective values, and not on a simulation. In multi-objective optimization, fitness inheritance has shown to work well (see for example [9]-[11]) and this has motivated the use of the concept in this work.

The described method of considering surrogate imprecision automatically adapts to the quality of the surrogate. A larger error of the surrogate leads a higher degree of randomness in the offspring selection, and hence the less the risk that the search is misled by the surrogate evaluations. In the same way, the smaller the error of the surrogate, the more the surrogate will impact selections. In comparison with existing methods that augment uncertainty information into surrogate evaluations, such as Lower Confidence Bound, Probability of Improvement and Expected Improvement (all described in [12]), advantages of the proposed method include:

- Simple to understand and to implement
- Requires no user-defined parameters
- Does not include any expensive computations (i.e. integrals)
- Can handle an arbitrary number of objectives without performance degradation
- Independent of the surrogate modeling technique

III. BENCHMARK OPTIMIZATION

A set of guidelines about systematic development of test problems for multi-objective optimization was first proposed in [15]. Based on these guidelines, [16] suggest six benchmark functions that have been used extensively in the literature for the analysis and comparison of multi-objective

EAs: ZDT1, ZDT2, ZDT3, ZDT4, ZDT5, and ZDT6. The features of these problems represent aspects that are known to cause difficulties in converging to the true Pareto-optimal front, and they reflect properties of real-world problems (such as multimodality and non-separability). This has motivated the use of these functions in assessing the performance of MOPSA-EA. However, function ZDT5 has been omitted since it defines a Boolean function over binary-strings, and such binary encoded solutions are not relevant in this study.

The metrics used to assess the performance of MOPSA-EA on the benchmark functions are described in the next section. The configuration of the surrogate and the algorithm parameters settings adopted in the evaluation are presented in Section B and C, respectively. In Section D, three existing surrogate-assisted multi-objective algorithms used for comparison are outlined, followed by a presentation of results in Section E.

A. Evaluation Metrics

In multi-objective optimization, there are two overall goals: convergence to the Pareto-optimal front, and maximal diversity among Pareto-optimal solutions. Two commonly used measures for evaluating convergence and diversity, respectively, for problems having a known true Pareto-optimal front are the Y and Δ metrics [13]. The Y metric measures the degree of convergence by calculating minimum Euclidean distances from each of the obtained non-dominated solutions to the closest solution in the true Pareto front. The smaller the value of Y, the better the convergence of the algorithm. The Δ metric measures the spread among solutions in the obtained non-dominated set using the following formula:

$$\frac{d_f + d_l + \sum_{i=1}^{N} |d_i - \bar{d}|}{d_f + d_l + (N-1)\bar{d}}.$$

In this formula, d_f and d_l are the Euclidean distances between the extreme solutions of the true Pareto front and the boundary solutions of the obtained non-dominated set, and $\bar{d}$ is the average of all distances d_i in the obtained non-dominated set ($N-1$ distances with N solutions). The smaller the value of Δ, the better the spread of solutions.

A problem of using separate metrics for convergence and diversity, as done with the Y and Δ metrics, is that a comparative evaluation of two algorithms might not give an answer about which of the algorithms is superior. While the first algorithm may have a low Y-value and a high Δ-value, the inverse may apply for the second algorithm. To address this problem, we suggest a new performance metric for problems having a known true Pareto front that combines convergence and diversity into a single measure. This metric, called Ω, is calculated by taking the average of all Euclidean distances from each true Pareto front sample to the closest solution generated by the algorithm (Fig. 3). The rationale behind this metric is that for a low value of Ω, both a well spread front and a good convergence is needed.

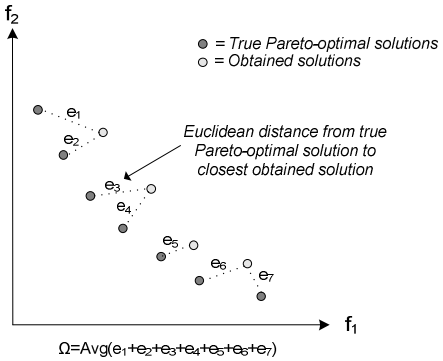


Fig. 3. Ω performance metric.

Another performance metric that combines both convergence and diversity is the $\mathcal{S}$ metric (also called the hyper-volume metric). Basically, $\mathcal{S}$ measures the volume in objective space dominated by obtained solutions (Fig. 4) [17]. The larger the volume, the better the results of the algorithm.

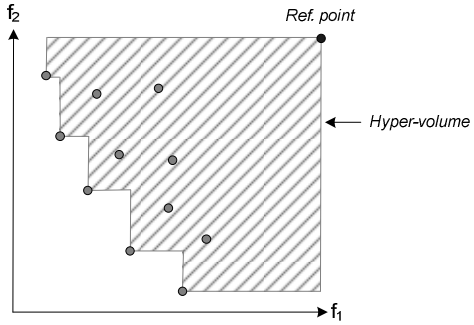


Fig. 4. $\mathcal{S}$ performance metric.

The $\mathcal{S}$ metric does not assume that the true Pareto-optimal front is known and can therefore also be applied to real-world problems. A potential problem of $\mathcal{S}$ is, however, that the measure is biased towards a diagonal front. This problem is illustrated in Fig 5. The solutions A and B both contribute to the Pareto front, but B is valued much higher than A since it contributes more to the hyper-volume.

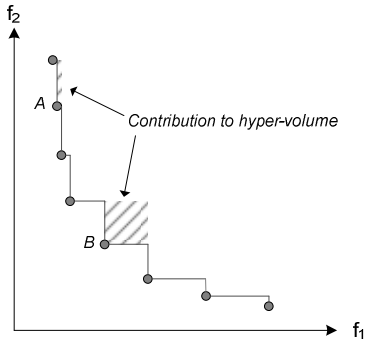


Fig. 5. Illustration of problem with $\mathcal{S}$ metric.

In the benchmark evaluation of MOPSA-EA, all four performance metrics described (Y, Δ, Ω, and $\mathcal{S}$) are being used for performance assessment.

B. Surrogate

MOPSA-EA allows for any kind of surrogates, and in this paper Artificial Neural Networks (ANNs) are being used since ANNs have been considered being appropriate for approximation of complex problems with limited number of data samples [6], [18]. The ANN adopted has a feed-forward architecture with one hidden layer. It has been shown (e.g. [19]) that one hidden layer is sufficient for nearly all problems. The ANN is trained using back-propagation with a learning rate of 0.5. For each 10^{th} simulation, the ANN is re-trained with the most recent samples (at most 50). The idea of regularly re-training the ANN with the most recent samples is to have a local surrogate defined over the current search region. Local ANNs have been preferred over global ANNs in surrogate-assisted optimization [20], mainly because they reduce the time-consumption of the training process [21]. To avoid overfittning, 10-folded cross-validation is used in the training.

The number of hidden nodes of the ANN is dynamically adapted to the number of samples available. For a good performance of an ANN, it is recommended that the number of weights of the network is proportional to the size of the training data set [22]. Since the number of samples continuously increase during the optimization, a static number of hidden nodes is not appropriate. Therefore, the optimization starts with an ANN of one single node, and additional hidden nodes are successively being added. When the number of samples available exceeds five times the number of weights in the network, a new hidden node is added (according to the weight-sample ratio suggested in [22]).

C. Algorithm Parameter Settings

The performance of an EA is highly dependent on the proper settings of its parameters, which are problem-dependent and usually must be obtained through trial-and-error experiments. Therefore, the parameter values of MOPSA-EA are tuned before the algorithm is actually being evaluated. Three different settings are being tested for each parameter:

- Population size: 20, 60, and 100.
- Number of offspring: 20, 60, and 100.
- Mutation step size: 0.5, 1.0, and 1.5.
- Crossover: Single-point (SP), blend, and uniform.
- Crossover probability: 0.4, 0.6, and 0.8.

All combinations of the different settings are tested for all parameters, which mean that in total 3^5 experiments are performed for each of the five benchmark functions. Each experiment is replicated 100 times and the average values of the Y, Δ, Ω, and $\mathcal{S}$ metrics are taken as the result. In finding the optimal parameter settings for a function, metric values from all experiments are collected and ranked according to their achieved values. By taking the sum of the achieved ranks of each metric, the best setting is identified. The optimal settings found for the five functions are presented in Table I.

TABLE I
OPTIMAL PARAMETER SETTINGS FOR MOPSA-EA

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
Population size	20	20	60	20	20
No. of offspring	60	100	60	100	60
Mutation step size	0.5	0.5	0.5	0.5	1.0
Crossover	SP	SP	SP	SP	SP
Crossover prob.	0.8	0.6	0.8	0.8	0.8

D. Performance Comparison

To assess the relative performance of MOPSA-EA, it is compared to three existing surrogate-assisted multi-objective algorithms, namely "Metamodel-Assisted $\mathcal{S}$ Metric Selection Evolutionary Multi-Objective Algorithm" (SMS-EMOA) [17], "$(\mu+\nu<\lambda)$ Metamodel-Assisted Evolution Strategy (MAES)" incorporated into NSGA-II [12], and NSGA-II integrated with an ANN (NSGA-II-ANN) [23].

SMS-EMOA (described with pseudo code in Fig. 6) is a steady-state algorithm that uses the $\mathcal{S}$ metric as selection criterion, both for offspring selection and replacement selection in the population. In both selections, a non-dominated sort takes place and the solution contributing most (in the former) or least (in the latter) to the hyper-volume of the population is selected.

```
P ← Create( )
Simulate(P)
while (not Stop_Optimization( )) do
  O ← ∅
  parent1 ← Select(P)
  parent2 ← Select(P)
  repeat λ times
    o ← Crossover(parent1, parent2)
    o ← Mutate( )
    O.Add(o)
  end
  SurrogateEvaluation(O)
  q ← SelectBest(O)
  Simulate(q)
  P.Add(q)
  P.Remove(SelectWorst(P))
endwhile
```

Fig. 6. Metamodel-Assisted $\mathcal{S}$ Metric Selection Evolutionary Multi-Objective Algorithm (SMS-EMOA).

MAES (described with pseudo code in Fig. 7) also uses the $\mathcal{S}$ metric for selections, but is based on a generational approach. From the population of μ solutions, λ offspring are generated and evaluated using the surrogate. Out of the λ offspring, the ν solutions contributing most to the hyper-

volume of the population is selected and simulated. The next generation of the population is then formed from the combined set of μ parents and ν offspring.

```
P ← Create( )
Simulate(P)
while (not Stop_Optimization( )) do
  O ← ∅
  repeat λ times
    o ← Crossover(Select(P),Select(P))
    o ← Mutate( )
    O.Add(o)
  end
  SurrogateEvaluation(O)
  Q ← Select Best(O)
  Simulate(Q)
  P ← Select(P ∪ Q)
endwhile
```

Fig. 7. $(\mu+\nu<\lambda)$ Metamodel-Assisted Evolution Strategy (MAES).

NSGA-II-ANN (described with pseudo code in Fig. 8) works like the standard NSGA-II, except that a simulation and an ANN is used alternately to evaluate generations. In every cycle of *m* generations, the simulation is first used to evaluate *n* of the generations and the surrogate is then used to evaluate the remaining *m-n* generations. A new surrogate is constructed in every cycle based on the last *n* simulation samples. Similar to MOPSA-EA, the idea is to adopt local surrogates defined over a small search region.

```
P ← Create( )
Simulate(P)
numEvaluations ← 0
simulate ← true
while (not Stop_Optimization( )) do
  O ← ∅
  repeat λ times
    o ← Crossover(Select(P),Select(P))
    o ← Mutate( )
    O.Add(o)
  end
  if simulate
    Simulate(O)
  else
    SurrogateEvaluation(O)
  endif
  numEvaluations ← numEvaluations +1
  if numEvaluations > n
    simulate ← false
  else if numEvaluations > m
    simulate ← true
    numEvaluations ← 0
  endif
  P ← Select(P ∪ O)
  endwhile
```

Fig. 8. NSGA-II integrated with an ANN (NSGA-II-ANN).

For a fair performance comparison, tuning of parameter settings have been applied also to SMS-EMOA, MAES, and NSGA-II-ANN. In Table II-IV, the optimal parameter settings found for these algorithms for the five benchmark functions are shown. Note that NSGA-II-ANN does not make use of offspring candidates and therefore the parameter "number of offspring" does not apply to this algorithm. Instead, this algorithm uses the parameters *m* and *n*, which are set to 13 and 3 (respectively), according to the recommendations in [12]. The surrogate configuration of SMS-EMOA, MAES, and NSGA-II-ANN is the same as for MOPSA-EA.

TABLE II
OPTIMAL PARAMETER SETTINGS FOR SMS-EMOA

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
Population size	20	20	20	20	60
No. of offspring	60	100	60	100	60
Mutation step size	1.5	1.5	1.5	1.5	1.5
Crossover	SP	Blend	SP	SP	SP
Crossover prob.	0.8	0.6	0.8	0.8	0.6

TABLE III
OPTIMAL PARAMETER SETTINGS FOR MAES

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
Population size	100	100	100	100	100
No. of offspring	100	100	100	100	60
Mutation step size	0.5	0.5	0.5	0.5	0.5
Crossover	SP	Blend	SP	SP	SP
Crossover prob.	0.4	0.4	0.8	0.8	0.6

TABLE IV
OPTIMAL PARAMETER SETTINGS FOR NSGA-II-ANN

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
Population size	60	60	60	100	100
Mutation step size	0.5	0.5	0.5	0.5	0.5
Crossover	Blend	Blend	SP	Blend	Blend
Crossover prob.	0.6	0.4	0.8	0.6	0.4

E. Results

Results from the benchmark functions are shown in Table V. The optimization is performed for 3000 function evaluations and the results presented are an average of 300 replications. All results have a confidence probability of 0.99 or more, calculated using Welch's t-test (defined in [24]). Note that a low value of Y, Ω and Δ, and a high value of $\mathcal{S}$ is desirable. In calculating the first three metrics, a set of 500 uniformly-distributed solutions of the true Pareto front is derived.

In the table, alongside each metric value, a rank (R) is provided stating the relative results of the four algorithms. Taking the sum of ranks of the metrics on all functions shows that MOPSA-EA obtains the best results concerning the Y, Ω and $\mathcal{S}$ metrics. On the Δ metric, MOPSA-EA and SMS-EMOA achieves the same total rank. Interestingly, MOPSA-EA obtains better overall results than both SMS-EMOA and MAES on the $\mathcal{S}$ metric, even though these algorithms are explicitly designed to maximize this metric.

TABLE V
BENCHMARK RESULTS

	Υ	R	Δ	R	Ω	R	$\mathcal{S}$	R
ZDT1								
MOPSA-EA	0.066	1	0.883	1	0.056	1	0.962	1
SMS-EMOA	2.347	4	0.898	2	1.844	4	0.687	4
MAES	0.103	2	0.975	4	0.085	2	0.957	2
NSGA-II-ANN	1.364	3	0.939	3	0.975	3	0.82	3
ZDT2								
MOPSA-EA	0.112	1	1.009	3	0.576	1	0.872	1
SMS-EMOA	2.073	4	0.847	1	1.269	4	0.707	4
MAES	0.21	2	1.006	2	0.736	2	0.85	2
NSGA-II-ANN	1.006	3	1.039	4	1.065	3	0.792	3
ZDT3								
MOPSA-EA	0.016	1	0.842	1	0.188	1	0.981	1
SMS-EMOA	2.073	4	0.847	2	1.269	4	0.707	4
MAES	0.054	2	1.04	4	0.201	2	0.975	2
NSGA-II-ANN	0.449	3	0.927	3	0.259	3	0.926	3
ZDT4								
MOPSA-EA	21.86	1	1.102	3	5.569	2	0.975	2
SMS-EMOA	66.76	4	0.94	1	60.98	4	0.748	4
MAES	28.15	2	1.14	4	4.57	1	0.98	1
NSGA-II-ANN	29.57	3	1.085	2	10.64	3	0.956	3
ZDT6								
MOPSA-EA	2.712	2	0. 93	1	2.52	1	0.728	1
SMS-EMOA	5.659	4	0.958	3	5.333	4	0.712	4
MAES	2.824	3	0.974	4	2.691	3	0.7274	3
NSGA-II-ANN	2.498	1	0.95	2	2.591	2	0.7279	2
Total								
MOPSA-EA		6		9		6		6
SMS-EMOA		20		9		20		20
MAES		11		18		10		10
NSGA-II-ANN		13		14		14		14

In the next section, the evaluation of MOPSA-EA on a real-world optimization problem from the manufacturing domain is presented.

IV. REAL-WORLD OPTIMIZATION

To evaluate the industrial strength of MOPSA-EA, it is applied to a real-world optimization problem. The problem considered concerns optimal production planning at a Volvo Aero factory in Sweden. The factory produces engine components to civilian and military airplanes, as well as to space rockets. Recently, a new manufacturing cell has been introduced that processes a wide range of different engine components.

The inflow of the cell is controlled by using fixed inter-arrival times of components. The inter-arrival time does not only specify when a component should enter in the system, but it also determines the component's due date since an overall production strategy is to process no more than one component of a specific type in the cell at a time. For an efficient production, the inter-arrival times should be specified in a way that maximizes the utilization of the cell and simultaneously minimizes overdue components (i.e. tardiness). For a high utilization, short inter-arrival times are needed in order to obtain a high load of the cell and thereby avoid machine starvation. However, avoiding overdue components requires generous due dates; that is long inter-arrival times. This means that the two objectives of maximal utilization and minimal tardiness are conflicting with each other.

The optimization reported in this paper considers a production scenario comprising eleven different component types. A simulation model of the manufacturing cell is built using the SIMUL8 software package. As a fast surrogate of the simulation, an ANN is constructed. The ANN has eleven input nodes (each of them corresponds to the inter-arrival time of a specific component type), and two output nodes (corresponding to utilization and tardiness, respectively). The basic configuration of the ANN with respect to topology and training procedure is the same as for the ANN used in the benchmark optimizations (described in Section III-B). Regarding the configuration of the algorithm parameters, the population size is set to 40, the number of offspring is set to 20, the mutation step size is set to 1.0, and single-point crossover is being used with a probability of 0.8.

In Table VI, results from the real-world problem are presented (average of 10 replications). The optimization is performed for 400 simulations, which is the maximum number of simulations that can be performed within the optimization time-budget defined by the company. The results presented have a confidence probability of at least 0.8 (calculated using Welch's t-test). Note that since the true Pareto-optimal front of this problem is unknown, only the $\mathcal{S}$ metric is calculated. As shown in the table, MOPSA-EA achieves the best $\mathcal{S}$ value and SMS-EMOA the worst, similar to the results of the benchmark functions.

TABLE VI
REAL-WORLD OPTIMIZATION

	$\mathcal{S}$
MOPSA-EA	0.465
SMS-EMOA	0.374
MAES	0.426
NSGA-II-ANN	0.435

In the next section, general conclusions from the study are presented.

V. CONCLUSIONS AND FUTURE WORK

In this paper, a new multi-objective EA for solving computationally-intensive optimization problems has been described. This algorithm is relatively simple in its design and implementation. For example, only one population is maintained, unlike several other multi-objective EAs which maintain sub-populations or keep track of an external archive of solutions. Furthermore, in contrast to many other surrogate-assisted algorithms, there is a minimum number of user-defined parameters related to the surrogate usage. Only one parameter, the size of the offspring pool, needs to be specified.

The proposed algorithm also defines an efficient approach of integrating surrogates. Contrary to the many other surrogate-assisted multi-objective EAs (e.g. [20], [25]-[26]), simulation evaluations are not compared with surrogate evaluations. The main problem of such comparison is that surrogate fitness and simulation fitness are treated interchangeably in the evolutionary operators, which are likely to lead to poor performance in complex problems having surrogates associated with a high degree of

imprecision [6]. In addressing the problem of uncertainties in surrogate predictions and to ensure a good convergence, the proposed algorithm adopts a new method for multi-objective optimization. Contrary to previous methods of handling surrogate imprecision, this method is parameterless, generally applicable to all types of surrogate techniques, easy to understand and implement, without expensive computations, and scalable with respect to the number of objectives.

The next step of improving the algorithm will be to extend it to handle simulation noise. To capture the stochastic behavior of complex real-world systems, simulations contain randomness. Instead of modeling only a deterministic path of how the system evolves in the process of time, a stochastic simulation deals with several possible paths based on random variables in the model. To tackle the problem of noise in output samples is crucial because the normal path of the algorithm would be severely disturbed if estimates of the objective function come from only one simulation replication. The common technique to handle noise is to send the algorithm with the average values of output samples obtained from a large number of replications. Although this technique is easy to implement, the large number of replications needed to obtain statistically confident estimates from computationally expensive simulation models of complex systems can easily render the approach to be totally impractical. Finding efficient methods to address the problem of noise is an important topic for further research [27], especially in multi-objective optimization [21].

ACKNOWLEDGMENT

The authors would like to thank Professor Kalyanmoy Deb for his valuable comments on this study.

REFERENCES

[1] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons Ltd, 2004.

[2] Y.S. Ong, P.B. Nair, A.J. Keane, and K.W. Wong, "Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems," in *Knowledge Incorporation in Evolutionary Computation*, Springer, 2004, pp. 307-332.

[3] H. Ulmer, F. Streichert, and A. Zell, "Evolution strategies assisted by gaussian processes with improved pre-selection criterion," in *Proc. IEEE Congress on Evolutionary Computation*, Canberra, Australia, December 8-12, 2003, pp. 692-699.

[4] J. Boesel, R.O. Bowden, F. Glover, J.P. Kelly, and E. Westwig, "Future of simulation optimization," in *Proc. Winter Simulation Conference*, Arlington, VA, USA, December 9-12, 2001, pp. 1466-1470.

[5] D. Chafeka, J. Xuan, and K. Rasheed, "Constrained multi-objective optimization using steady state genetic algorithms," in *Proc. Genetic and Evolutionary Computation Conference*, Springer-Verlag Germany, 2003, pp. 813-824.

[6] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing*, vol. 9, pp. 3-12, Springer-Verlag Germany, 2005.

[7] T. Lacksonen, "Empirical comparison of search algorithms for discrete event simulation," in *Computers & Industrial Engineering*, vol. 40, no. 1-2, 2001, pp. 133-148.

[8] M. Emmerich, A. Giotis, M. Özdemir, T.H. Bäck, and K. Giannakoglou, "Metamodelassisted evolution strategies," in *Proc. International Conference on Parallel Problem Solving from Nature*, Granada, Spain, September 7-11, 2002, pp. 361-370.

[9] M. Reyes-Sierra and C.A. Coello Coello, "Fitness inheritance in multi-objective particle swarm optimization," in *Proc. IEEE Swarm Intelligence Symposium 2005*, Pasadena, California, June 8-10, pp. 116-123, 2005.

[10] J.H. Chen, D.E. Goldberg, S.Y. Ho and K. Sastry, "Fitness Inheritance In Multi-objective Optimization," in *Proc. Genetic and Evolutionary Computation Conference*, pp. 319-326, 2002.

[11] L.T. Bui, H.A. Abbass and D. Essam, "Fitness inheritance for noisy evolutionary multi-objective optimization", in *Proc. Conference on Genetic and Evolutionary Computation*, Washington DC, USA, pp. 779-785, 2005.

[12] M. Emmerich, K. Giannakoglou, and B. Naujoks, "Single- and multi-objective evolutionary optimization assisted by gaussian random field metamodels," in *IEEE-TEC Special Issue on Evolutionary Computation in the presence of uncertainty*, vol. 10, no. 4, pp. 421-439, 2006.

[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multi-objective genetic algorithm-NSGA-II," KanGAL Report 2000001, Indian Institute of Technology Kanpur, 2000.

[14] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), May 2001.

[15] K. Deb, "Multi-objective genetic algorithms: Problem difficulties and construction of test functions," in *Evolutionary Computation*, vol. 7, pp. 205-230, 1999.

[16] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," in *Evolutionary Computation*, vol. 8, no. 2, 2000.

[17] M. Emmerich, N. Beume, and B. Naujoks, "An EMO algorithm using the hypervolume measure as selection criterion," in *Proc. Evolutionary Multi-Criterion Optimization*, Springer Berlin-Heidelberg, 2005, pp. 62-76.

[18] A. Ratle, "Optimal sampling strategies for learning a fitness model," in *Proc. Congress Evolutionary Computation*, Washington DC, USA, July 6-9, 1999, pp. 2078–2085.

[19] T. Chen, H. Chen, and R. Liu, "Approximation capability in $\{C(R^n)\}$ by multilayer feedforward networks and related problems," in *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 25-30, 1995.

[20] A.P. Giotis, K.C. Giannakoglou J., and Periaux, "A reduced-cost multi-objective optimization method based on the pareto front technique, neural networks and PVM," in *Proc. European Congress on Computational Methods in Applied Sciences and Engineering*, Barcelona, Spain, September 15-17, 2000.

[21] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments – a survey," in *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303-317, 2005.

[22] K. Mehrotra, C.K. Mohan, and S. Ranka, *Elements of Artificial Neural Networks*, MIT Press, 1996.

[23] P.K.S. Nain and K. Deb, "A multi-objective optimization procedure with successive approximate models," KanGAL report no. 2005002, Indian Institute of Technology, Kanpur, India, 2005.

[24] A.M. Law and D. Kelton, *Simulation Modeling and Analysis*, Mc Graw Hill, 3rd edition, 2000.

[25] I.I. Voutchkov and A. J. Keane, "Multiobjective optimization using surrogates," in *Proc. 7th International Conference on Adaptive Computing in Design and Manufacture*, Bristol, 2006, pp. 167-175.

[26] M.K. Karakasis and K.C. Giannakoglou, "Metamodel-assisted multi-objective evolutionary optimization," *Proc. Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems*, Munich, Germany, September 12-14, 2005.

[27] A. Gosavi, *Simulation-based optimization: parametric optimization techniques and reinforcement learning*. Boston, Mass., Kluwer Academic, 2003.

Multi-Objective Simulation-Based Optimization of Production Systems with Consideration Noise

Anna Syberfeldt[1], Henrik Grimm[1], Amos Ng[1]
[1]Center for Intelligent Automation, University of Skövde, Skövde, Sweden
anna.syberfeldt@his.se

ABSTRACT

Many production optimization problems approached by simulation are subject to noise. When evolutionary algorithms are applied to such problems, noise during evaluation of solutions adversely affects the evolutionary selection process and the performance of the algorithm. In this paper we present a noise compensation technique that efficiently deals with the negative effects of noisy simulations in multi-objective optimization problems. Basically, this technique uses an iterative re-sampling procedure that reduces the noise until the likelihood of selecting the correct solution reaches a given confidence level. The technique is implemented in MOPSA-EA, an existing evolutionary algorithm designed specifically for real-world simulation-optimization problems. In evaluating the new technique, it is applied on a benchmark problem and on two real-world problems of manufacturing optimization. A comparison of the performance of existing algorithms indicates the potential of the proposed technique.

Keywords: simulation, optimization, noise.

1. INTRODUCTION

Real-world production problems often contain nonlinearities, combinatorial relationships and uncertainties that are too complex to be modelled analytically. In these scenarios, simulation-based optimisation (SO) is a powerful tool to determine optimal system settings [1]. SO is the process of finding the best values of some parameters for a system, where the performance of the system is evaluated based on the output from a simulation model of the system. In a production system, for example, one might be interested in finding the optimal buffer settings with respect to throughput of the system and average cycle time of products. Finding the optimal parameter values is an iterative simulation-optimisation process. An optimisation procedure generates a set of parameter values and feeds them to a simulation that estimates the performance of the system. Based on the evaluation feedback from the simulation, the optimisation procedure generates a new set of parameter values and the generation-evaluation process continues until a user-defined stopping criterion is satisfied.

While traditional analytical optimization methods have been unable to cope with the challenges imposed by many real-world production problems approached by simulation, such as multimodality, non-separability and high dimensionality, evolutionary algorithms (EAs) have proven to be powerful in searching reasonably good solutions. EAs are also particularly suitable to solve problems that require simultaneous optimization of more than one objective, which is often the case in real-world applications [2]. The difficulty with problems of multiple objectives is that there is usually no single optimal solution with respect to all objectives, as improving performance on one objective deteriorates performance of one or more of the other objectives. Instead of a single optimum, there is a set of optimal trade-offs between the conflicting objectives, known as the Pareto-optimal solutions or the Pareto-front. A solution is Pareto-optimal if it is not dominated by any other solution, i.e. there exist no other solution that is superior in one objective without being worse in another objective. Different ranks of domination can also be assigned, where rank 1 includes the Pareto-optimal solutions in the complete population, rank 2 the Pareto-optimal solutions identified when temporarily discarding all solutions of rank 1, etc. Since EAs maintain a population of solutions they can, contrary to many other optimization techniques, capture multiple Pareto-optimal solutions in one single optimization run.

In real-world simulation-optimization using EAs, one has not only to cope with multiple optimization objectives, but also with noise. In practical production problems, the existence of noise during evaluation of solutions is inevitable [3-4] For example, when trying to optimize the operation of an assembly machine by tuning machine control parameters, the outcome from different evaluations will not be identical even though all parameters have been fixed. The sources of noise causing the unpredictable outcome can be manifold, such for example human operators or worn-out parts of the machine. To capture the stochastic behaviour of systems like this, simulations contain randomness. Instead of modelling only a deterministic path of how the system evolves in the process of time, a stochastic simulation deals with several possible paths based on random variables in the model. In situations like these, improving the outcome can be a risky endeavour because one can never be sure that a seeming improvement obtained by a certain control parameter

change is a real improvement. If estimates of the objective function come from only one simulation replication the normal path of the EA can be severely disturbed with substantial performance degradation as a consequence. Since noisy fitness values are used for selection, the algorithm can be mislead to propagate inferior solutions; bad ones might be kept for the next generation, and the good ones might be excluded [4].

To address the problem of noise in simulations is critical for successful results in real-world production problems. Compared to its practical relevance, however, noise compensation has gained only limited attention in EA research [5], and especially in multi-objective optimization there are few studies devoted to optimization of noisy problems [4]. The common technique to handle noise is to send the algorithm with the average values of output samples obtained from a large number of simulation replications. Although this technique is easy to implement, the large number of replications needed to obtain statistically confident estimates from computationally expensive simulation models of complex systems can easily render the approach to be totally impractical.

In this paper we present new noise compensation technique for evolutionary selection in multi-objective problems that the efficiently deal with negative effects of noise in simulations. This technique uses an iterative re-sampling procedure that reduces the noise until the likelihood of selecting the correct solution reaches a given confidence level. The noise compensation technique is integrated in an existing EA called "Multi-Objective Parallel Surrogate-Assisted EA" (MOPSA-EA) (described in [6]). MOPSA-EA is designed to reduce the huge time-consumption associated with many simulation-based optimization problems. Real-world optimization problems often involve an immense number of possible solutions, and an EA requires a large number of simulations before an acceptable solution is be found [7-8]. This holds true especially in multi-objective problems, where a significantly larger portion of the search space needs to be explored to obtain the whole Pareto-optimal set [9]. Even with improvements in computer processing speed, one single simulation evaluation may take a couple of minutes to hours or even days of computing time [10]. To tackle the problem of efficiency, MOPSA-EA supports a high degree of parallelism by implementing the master-slave parallelization scheme in combination with a steady-state design. For improved efficiency, the algorithm also utilizes a simulation surrogate (also called metamodel). The surrogate is used to screen candidate solutions and identify the most promising one. Instead of generating only a single offspring, which is normally done in steady-state algorithms, a pool of multiple offspring is created. Each of the offspring is evaluated by the surrogate, and the best one is simulated and inserted into the population. In the next section, the details of MOPSA-EA are further described.

2. DESCRIPTION OF MOPSA-EA

In MOPSA-EA, initially, the first generation of the population P is filled with μ random solutions. While the population is not full and there are processing nodes available, new random solutions are being created and sent to the simulation for evaluation. When μ solutions have been simulated, offspring generation is initiated. Offspring are created from parents in P chosen using crowding tournament selection (described in [2]). With tournament selection, solutions with worse fitness may also be selected, which maintains diversity in the population and prevents premature convergence. For the tournament, two solutions A and B are chosen randomly and A is declared as the winner over B if either (*i*) A has a better rank than B, or (*ii*) A and B have the same rank, but A has a larger crowding distance than B.

When generating offspring, instead of creating only a single new solution from a pair of parents, which is the commonly used strategy in steady-state algorithms, a pool of λ candidate offspring is created. Such an offspring pool, called O, is created as soon as a processing node becomes available. The solutions in O are evaluated by the surrogate, and since the computational cost of surrogate evaluations can be neglected in real-world optimizations [11], the size of the pool might be large. The surrogate objective values assigned to solutions in O are adjusted to take the imprecision of the surrogate into consideration. This is done by modifying the values based on the calculated error of the surrogate. For each offspring, the objective errors of its parents are calculated by evaluating the parents using the surrogate and taking the difference between their assigned simulation objective values and the obtained surrogate objective values. Since the accuracy of the surrogate might change dynamically, surrogate values are calculated every time a solution is chosen as parent. The surrogate objective values of an offspring are modified by adding the weighted mean of the error values of the offspring's parents. The weighting is based on each parent's influence on the child during crossover. In case no crossover is applied when creating the child (i.e. only mutation is performed), the influence of one of the parents is 100%. The described method of considering surrogate imprecision automatically adapts to the quality of the surrogate. A larger error of the surrogate leads a higher degree of randomness in the offspring selection, and hence the less the risk that the search is misled by the surrogate evaluations. In the same way, the smaller the error of the surrogate, the more the surrogate will impact selections.

Based on the adjusted surrogate objective values, the most promising solution in O to insert into P is selected. In this procedure, all solutions of rank 1 in O are identified (called O_{R1}) and checked for domination against all solutions of rank 1 in P (called P_{R1}). By only identifying O_{R1} and P_{R1}, a full non-dominating sort is

avoided. The solution in O_{R1} dominating most solutions in P_{R1} is selected and simulated. If several solutions in O_{R1} share the position of dominating most solutions in P_{R1}, the one having the largest Euclidean distance to its closest neighbor in P_{R1} is selected (note that crowding distance cannot be used since the solutions of O_{R1} and P_{R1} might be of different ranks if merged). Before the selected offspring is inserted into P, the worst solution in P is removed by performing a non-dominated sort and discarding the solution with the smallest crowding distances in the last rank. An elitistic approach is also possible, in which an offspring is only inserted into P if it is not dominated by all solutions in P. The simulation sample obtained from a newly inserted offspring may be used to update the surrogate. To save time, an update does not need to take place every time a new sample becomes available, but only every *N*:th sample.

When noise during evaluation of solutions is present, this adversely affects the parental selection process and the performance of the algorithm. If this is not considered, the algorithm can be mislead to propagate inferior parents. In the next section, we present a new technique for reducing noise to be used in the parental tournament selection in MOPSA-EA.

3. A MODIFIED EVOLUTIONARY SELECTION OPERATOR CONSIDERING NOISE

In real-world problems, the characteristics of the noise are unknown and re-sampling of solutions is therefore always necessary to form an estimate of the noise size. However, since sampling *n* times increases the computational time by a factor of *n*, it is important that the number of samplings is kept at a minimum. Many noise handling methods use an approach of re-sampling solutions a fixed number of times in each generation, which is inefficient from two perspectives: (*a*) the noise size might vary in the search space, and (*b*) simulations are allocated in a suboptimal way (efforts are wasted on inferior solutions, and solutions subject to less noise are sampled unnecessarily and those suffering from much noise are not sampled enough). For improved efficiency, we therefore introduce a technique that vary the number of samples used per solution based on the present noise size in combination with the required confidence level, representing how certain we want to be to choose the better of two solutions. Re-sampling of solutions is performed iteratively until the noise is sufficiently reduced. We refer to this technique as *confidence-based dynamic re-sampling*. Basically, the technique is implemented by using an iterative re-sampling procedure that continues sampling two solutions until the likelihood of selecting the correct solution reaches the given confidence level. Below, the procedure of the proposed confidence-based dynamic re-sampling technique is described in further detail:

Step 1: Initial sampling

Initially, the two solutions being compared are sampled two times each, which is the minimum number of samples needed to form an initial estimate of the present noise size.

Step 2: Calculation of mean and sample standard deviation

Based on the collected samples, the mean μ and sample standard deviation s of each objective of the two solutions are calculated. The sample standard deviation, measuring the variability in samples (i.e. the noise size), of objective i is calculated according to Equation 1

$$s_i = \sqrt{\frac{1}{N-1}\sum_{j=1}^{N}\left(x_j - \mu_i\right)^2} \qquad (1)$$

where N is the number of samples, and $x_1, ..., x_N$ are the sample values.

Step 3: Selection of confidence level

The confidence level is defined by the user and represents the required certainty of the relation between two solutions. Between two solutions A and B, three types of relations are possible: (*i*) A dominates B, or (*ii*) B dominates A, or (*iii*) A and B are mutually non-dominating.

The confidence level α specifies that in at least α of the cases the selection between two solutions should be correct. One confidence level is defined for each Pareto-rank, and generally the higher the rank of a solution, the higher its confidence level. This is because a high precision is usually more important for solutions in, or nearby, the Pareto-front. The confidence level to use in a comparison of two solutions is derived from the one with highest rank, which means that a non-dominating sort must be performed in this step to derive the ranks of the solutions.

In this study, the following confidence levels are being defined:

Rank 1: 0.75

Rank 2: 0.70

Rank 3: 0.65

Rank 4: 0.60

Rank 5 and higher: 0.55

Step 4: Confidence test

In a noisy context, the true relation between two solutions is only possible to determine by taking the mean of all possible samples of the solutions. In reality, however, it is not possible to collect the complete set of samples, but only a limited number of samplings can be performed. Instead, the probability that the solutions' relation is the same given the collected samples as given all samples has to be established (i.e. the

probability of making a correct selection between the solutions). In doing this, the method of Welch Confidence Interval (WCI) is being used. WCI can be used in comparing whether or not there is a significant difference between two solutions of unknown and possibly unequal variances with respect to a given confidence level.

WCI values are calculated for each objective i according to Equation 2 [12]

$$\mu_{iA}(N_A) - \mu_{iB}(N_B) \pm t_{f,1-\frac{\alpha}{2M}} \sqrt{\frac{s_{iA}^2}{N_A} + \frac{s_{iB}^2}{N_B}} \qquad (2)$$

where A and B are the two solutions being compared, μ is the mean of objective i, N is the number of samplings of a solution, s^2 is the variance of objective i (Equation 1), and f is the estimated degree of freedom (Equation 3), α is the confidence level, and M is the number of objectives. In a multi-objective problem, the confidence level α has to be divided by 2 M (and not by 1, as normally done) due to the Bonferroni Inequality [12]. This means that for a problem of two objectives, to obtain a 0.95 confidence each objective has to be compared with a confidence level of 0.975.

$$f = \frac{\left[\frac{s_A^2}{N_A} + \frac{s_B^2}{N_B}\right]}{\frac{\frac{s_A^2}{N_A}}{N_A - 1} + \frac{\frac{s_B^2}{N_B}}{N_B - 1}} \qquad (3)$$

If the WCI resulting from Equation 3 does not cover 0, there is a significant difference between the two solutions in the *i*:th objective. If none of the WCIs for the *M* objectives cover 0, it means that the relation between the two solutions (see Step 3) can be established with respect to the given confidence level. The dominating solution is then returned, or the one with largest crowding distance if the solutions are mutually non-dominating, and the procedure is terminated. Otherwise, i.e. if any of the objectives' intervals cover 0, the difference between the solutions is not significant and further noise reduction is necessary in order to determine their internal relation.

Step 5: Noise reduction by re-sampling

When the given confidence level is not reached, additional re-sampling is required. Ultimately, the confidence level should be reached using as few re-samplings as possible to save resources. Therefore, the strategy adopted in this step is to resample only one of the solutions, and select the one having the largest sample standard deviation in the objective with the largest overlapping intervals (i.e. with the largest potential to eliminate the undesired overlap). When a re-sampling has been performed, the procedure is repeated from step 2 and a new check is made if yet another sampling is necessary.

To prevent two solutions that are close to each other in objective space to be re-sampled forever, the number of times a solution can be sampled is limited. Similar to the specification of confidence level, the maximum number of samplings is defined by the user for each rank. A larger number of samplings is usually allowed for solutions in higher ranks where a higher precision is needed. The limited number of samplings means that if a solution in this step has already reached its allowed number, it cannot be further sampled. The other solution is then re-sampled instead, unless it has also reached its maximum number. In such case the dominating solution (or the one with largest crowding distance, if the solutions are mutually non-dominating) is returned and the procedure is terminated. In this study, the following maximum number of samplings is being defined:

Rank 1: 5 sampling

Rank 2: 4 sampling

Rank 3 and higher: 3 samplings

Since the ranks of solutions is calculated in every iteration of the procedure, the maximum number of samplings of solutions, as well as the confidence level to use when comparing them, may change between iterations. In this way, the re-sampling strategy becomes dynamic and automatically adjusted to the current situation.

In the following sections, an evaluation of the confidence-based dynamic re-sampling technique is presented when integrated in MOPSA-EA When MOPSA-EA is implemented to handle noise using this technique, the algorithm is called “N-MOPSA-EA”.

4. OPTIMIZATION PROBLEMS

Three optimization problems are used to assess the performance of N-MOSPA-EA; one theoretical benchmark problem and two real-world problems from the manufacturing domain.

Problem 1: Benchmark problem

The function “ZDT1”, described in Table 1, is a multi-objective benchmark problem used in many research articles. In this study, artificial noise is added to the function generated from a Gaussian distribution with $\mu = 0$ and σ representing the noise size. Three different noise sizes are being tested: 0.10, 0.15, and 0.20.

Table 1: ZDT1 benchmark function

No. of inputs	Input bounds	Function	Optimal solution	Shape
30	[0,1]	$f_1(x) = x_i$ $f_2(x) = g(x)\left[1 - \sqrt{x_1 / g(x)}\right]$ $g(x) = 1 + 9\left(\sum_{i=2}^{n} x_i\right)/(n-1)$	$x \in [0,1]$ $x_i = 0$ $i = 2,\ldots,n$	Convex Pareto-front

Problem 2: Engine Component Manufacturing

The first real-world problem considered concerns optimal production planning at a Volvo Aero factory in Sweden. The factory produces engine components to civilian and military airplanes, as well as to space rockets. The focus of the study is a manufacturing cell that processes a wide range of different engine components. The inflow of the cell is controlled by using fixed inter-arrival times of components. The inter-arrival time does not only specify when a component should enter in the system, but it also determines the component's due date since an overall production strategy is to process no more than one component of a specific type in the cell at a time. For an efficient production, the inter-arrival times should be specified in a way that maximizes the utilization of the cell and simultaneously minimizes overdue components (i.e. tardiness). For a high utilization, short inter-arrival times are needed in order to obtain a high load of the cell and thereby avoid machine starvation. However, avoiding overdue components requires generous due dates; that is long inter-arrival times. This means that the two objectives of maximal utilization and minimal tardiness are conflicting with each other.

Problem 3: Camshaft Machining Line

The second real-world problem considered is a camshaft machining line at Volvo Cars in Sweden. The line is responsible for producing about 15 different camshaft variants. The machining line consists of 14 different machine groups with one to seven parallel machines in each group; totally 34 machines. Unlike an ordinary flow shop with parallel machines, each machine has its own processing time, physical capability and limitations, as well as variability in terms of failures and set-ups. The machining line is semi-automated with robots that feed machines inside cells, while the loading and unloading of camshafts are performed by operators.

The processing to be performed in the line is given by a schedule, defining an internal priority order among batches of different product variants and the individual path through machines and operations these batches should take. To be able to handle demand fluctuations and unexpected events such as machine breakdowns or quality defects, it is important that the schedule is defined in a way that makes sure that the number of different camshaft variants in the finished goods stock is above the specified minimum levels. Stock levels are check at continuous time intervals, and a mean value of the shortage noticed at each measure point is calculated at the end of the scheduling period. Besides minimizing shortage, it is also important that schedule results in as high throughput of the line is as possible for maximum efficiency. For high throughput, there should ideally be only one variant produced in the line at the same time to avoid set-up times of machines, which cause a large overhead. Further, for a high throughput, variants with shorter processing times should be prioritized before those with longer processing times. However, to maintain the minimum stock levels, an even mix of the different variants being produced in the line is needed and variants should be prioritized in a way that avoids shortage. In other words, the objectives of minimum shortage and maximum throughput are conflicting.

5. EVALUATION

In assessing the performance of N-MOPSA-EA a number of evaluation metrics are being used, which are described in the next section. The configuration of the surrogates adopted in the optimization problems is presented in Section 5.2. In Section 5.3, three existing surrogate-assisted multi-objective algorithms used for comparison are outlined.

5.1 Evaluation metrics

An overall goal in multi-objective optimization is convergence to the Pareto-front. A commonly used measure for evaluating convergence for problems having a known true optimal front (which is the case with the ZDT function) is the Y metric ([13]. This metric measures the degree of convergence by calculating the average minimum Euclidean distances from each of the obtained non-dominated solutions to the closest solution in the true Pareto-front. The smaller the value of Y, the better the convergence of the algorithm. The Y metric is used in assessing the performance in the ZDT1 problem (it cannot be used in the other two problems since the true Pareto front of these problems is unknown, as with real-world problems in general). In the ZDT1 problem, the Ω metric is also used (described in [6]). This metric is a combined measure of convergence and diversity in the in the set of non-dominated solutions. Ω is calculated by taking the average of all Euclidean distances from each true Pareto-front sample to the closest solution generated by the algorithm. The lower the value of Ω, the better the results of the algorithm. Another performance metric that combines both convergence and diversity is the $\mathcal{S}$ metric (also called the hyper-volume metric). Basically, $\mathcal{S}$ measures the volume in objective space dominated by obtained solutions [14]. The larger the volume, the better the results of the algorithm. The $\mathcal{S}$ metric does not assume that the true Pareto-optimal front is known and can therefore also be applied to real-world problems.

In the evaluation of N-MOPSA-EA, all three performance metrics described (Y, Ω, and $\mathcal{S}$) are being used in the benchmark problem, while only the last metric ($\mathcal{S}$) is being used in the real-world problems.

5.2 Surrogates

MOPSA-EA allows for any kind of surrogates, and in this paper two different surrogate techniques are being used. In the first two problems, Artificial Neural Networks (ANNs) are being used. ANNs have been considered being appropriate for approximation of complex problems with limited number of data samples [15]. The ANN adopted has a feed-forward architecture with one hidden layer. The ANN is trained using back-

propagation with a learning rate of 0.5. For each 10th simulation, the ANN is re-trained with the most recent samples (at most 50). The idea of regularly re-training the ANN with the most recent samples is to have a local surrogate defined over the current search region. Local ANNs have been preferred over global ANNs in surrogate-assisted optimization, mainly because they reduce the time-consumption of the training process [3]. To avoid overfittning, 10-folded cross-validation is used in the training. The number of hidden nodes of the ANN is dynamically adapted to the number of samples available. For a good performance of an ANN, it is recommended that the number of weights of the network is proportional to the size of the training data set [16]. Since the number of samples continuously increase during the optimization, a static number of hidden nodes is not appropriate. Therefore, the optimization starts with an ANN of one single node, and additional hidden nodes are successively being added. When the number of samples available exceeds five times the number of weights in the network, a new hidden node is added (according to the weight-sample ratio suggested in [16]).

In the third problem, constructing a useful ANN is not possible since the number of simulation inputs is very large (>500). An ANN with over 500 inputs involve tens of thousands of network weights, or even more, and such network cannot perform well when the problem is complex and the number of data samples is limited. Therefore, we have constructed a so called "surrogate model" instead of an ANN in this problem. While an ANN treats the simulation as a black box, knowing nothing about its inner workings, a surrogate model treats the simulation as a white box and explicitly attempts to imitate its internals. The surrogate model is built in the C# programming language and solves the same problem as the simulation through a number of simplifications (for example, carts transporting camshaft between machines are not modelled). Since it is less complex than the simulation, it is also computationally cheaper and thereby serves the same purpose as an ANN.

5.3 Performance comparison

To assess the relative performance of N-MOPSA-EA, it is compared to the original MOPSA-EA and to three existing surrogate-assisted multi-objective algorithms, namely "Metamodel-Assisted $\mathcal{S}$ Metric Selection Evolutionary Multi-Objective Algorithm" (SMS-EMOA) [14], " $(\mu + \nu < \lambda)$ Metamodel-Assisted Evolution Strategy (MAES)" incorporated into NSGA-II [11], and NSGA-II integrated with an ANN (NSGA-II-ANN) [17].

SMS-EMOA is a steady-state algorithm that uses the $\mathcal{S}$ metric as selection criterion, both for offspring selection and replacement selection in the population. In both selections, a non-dominated sort takes place and the solution contributing most (in the former) or least (in the latter) to the hyper-volume of the population is selected.

MAES also uses the $\mathcal{S}$ metric for selections, but is based on a generational approach. From the population of μ solutions, λ offspring are generated and evaluated using the surrogate. Out of the λ offspring, the ν solutions contributing most to the hyper-volume of the population is selected and simulated. The next generation of the population is then formed from the combined set of μ parents and ν offspring.

NSGA-II-ANN works like the standard NSGA-II (see [13]), except that a simulation and an ANN is used alternately to evaluate generations. In every cycle of *m* generations, the simulation is first used to evaluate *n* of the generations and the surrogate is then used to evaluate the remaining *m-n* generations. A new surrogate is constructed in every cycle based on the last *n* simulation samples. Similar to MOPSA-EA, the idea is to adopt local surrogates defined over a small search region.

For a fair performance comparison of the five algorithms used in the evaluation, all algorithms start the optimization from the same initial population, use the same surrogate configuration, and have the same parameter settings (Table 2). Note, however, that NSGA-II-ANN does not make use of offspring candidates and therefore the parameter "number of offspring" does not apply to this algorithm. Instead, this algorithm uses the parameters *m* and *n*, which are set to 13 and 3 (respectively), according to the recommendations in [13].

Table 2: Algorithm Parameter Settings

	Problem 1	Problem 2	Problem 3
Population size	50	40	60
No. of offspring	25	20	30
Mutation step size	0.5	1.0	1.0
Crossover	Single-point	Single-point	Single-point
Crossover prob.	0.8	0.8	0.8

6. RESULTS

This section presents the results of the three optimization problems. In MOPSA-EA, SMS-EMOA, MAES and NSGA-II-ANN, two different strategies of sampling solutions are being tested: (*i*) one sampling of each solution (i.e. no noise reduction), and (*ii*) five samplings of each solution. With both strategies, the total number of simulation replications used in an optimization is the same. Consequently, the number of unique solutions evaluated will be different in the two strategies; given *n* simulation replications and *s* samplings, *n*/*s* unique solutions are being evaluated.

Problem 1: Benchmark problem

Results from the ZDT1 function are shown in Table 3-5. The optimization is performed for 5000 function evaluations and the results presented are an average of 500 replications. All results have a confidence probability of 0.99 or more, calculated using Welch's t-test (defined in [12]). Note that a low value of Y and Ω, and a high value of $\mathcal{S}$ is desirable. In calculating the first two metrics, a set of 500 uniformly-distributed solutions of the true Pareto front is derived. When calculating the performance metrics, the objective values *without* noise is used.

As shown from the tables, MOPSA-EA achieves the best results with noise size is 0.1, while N-MOPSA-EA achieves the best results with size 0.15 and size 0.2. This indicates that the original algorithm could be used when there is little noise present, and that N-MOPSA-EA is most efficient with more substantial noise.

MOPSA-EA, SMS-EMOA, MAES and NSGA-II-ANN all obtain better results when only performing one sampling compared to five. These results may indicate that the noise sizes used represent a relatively small amount of noise. This theory is supported by the fact that the benefit of performing replications generally seems to increase with the noise size.

Table 3: Results ZDT1 Noise Size 0.1

	Y		Ω		$\mathcal{S}$	
N-MOPSA-EA	0.887		0.825		0.84	
Samplings	*1*	*5*	*1*	*5*	*1*	*5*
MOPSA-EA	0.815	1.349	0.781	0.977	0.842	0.8
SMS-EMOA	2.052	2.278	1.577	1.931	0.708	0.657
MAES	1.094	1.932	1.065	1.823	0.794	0.696
NSGA-II-ANN	0.98	2.226	1.089	1.882	0.809	0.644

Table 4: Results ZDT1 Noise Size 0.15

	Y		Ω		$\mathcal{S}$	
N-MOPSA-EA	0.997		0.845		0.815	
Samplings	*1*	*5*	*1*	*5*	*1*	*5*
MOPSA-EA	1.001	1.455	1.22	1.249	0.782	0.752
SMS-EMOA	2.146	2.358	1.822	2.128	0.675	0.626
MAES	1.462	2.051	1.15	1.833	0.76	0.669
NSGA-II-ANN	1.285	2.092	1.24	1.829	0.761	0.664

Table 5: Results ZDT1 Noise Size 0.2

	Y		Ω		$\mathcal{S}$	
N-MOPSA-EA	1.147		0.943		0.802	
Samplings	*1*	*5*	*1*	*5*	*1*	*5*
MOPSA-EA	1.228	1.545	1.366	1.498	0.778	0.748
SMS-EMOA	2.219	2.427	1.677	1.919	0.702	0.664
MAES	1.694	2.206	1.477	1.992	0.727	0.681
NSGA-II-ANN	1.319	2.101	1.302	1.852	0.757	0.673

Problem 2: Engine Component Manufacturing

Results of the first real-world optimization problem are presented in Table 6 (average of 10 replications). The optimization is performed for 400 simulations. The results presented have a confidence probability of at least 0.8 (calculated using Welch's t-test). Unlike the theoretical ZDT1 problem, the true objective values cannot be derived in this problems to be used in calculating the performance of the algorithms. Instead, the final Pareto-front found by an algorithm are being replicated 20 times and the mean values of the simulation replications are used when calculating the $\mathcal{S}$ metric (note that all solutions in the front found by the algorithm might not end up in the Pareto-front when the mean values are considered).

As shown in Table 6, N-MOPSA-EA achieves the best results and SMS-EMOA the worst. In difference with the ZDT1 problem, all algorithms benefit from performing multiple samplings of solutions, which might indicate that the noise is stronger and has a more complicated nature in this problem compared to the artificial noise added to the ZDT1 function.

Table 6: Results Engine Component Manufacturing

	$\mathcal{S}$	
N-MOPSA-EA	0.496	
Samplings	*1*	*5*
MOPSA-EA-EA	0.465	0.467
SMS-EMOA	0.374	0.398
MAES	0.426	0.451
NSGA-II-ANN	0.446	0.452

Problem 3: Camshaft Machining Line

In this problem, the optimization is performed for 600 simulations and the results of the different algorithms are presented in Table 7. Similar to the previous problem, the $\mathcal{S}$ metric is calculated by replicating the final Pareto-front found by the algorithm 20 times and taking the mean values of the replications. The confidence probability of the results is at least 0.84 (calculated using Welch's t-test). As shown from the results, the algorithms rank in the same order as in the previous problem. Also in this problem all algorithms benefit from performing multiple samplings of solutions.

Table 7: Results Engine Component Manufacturing

	$\mathcal{S}$	
N-MOPSA-EA	0.511	
Samplings	*1*	*5*
MOPSA-EA	0.481	0.483
SMS-EMOA	0.408	0.44
MAES	0.441	0.469
NSGA-II-ANN	0.471	0.479

7. CONCLUSIONS

In this paper, a new technique that efficiently deals with the negative effects of noise in simulations is presented. Basically, this technique uses an iterative re-sampling procedure that reduces the noise until the likelihood of selecting the correct solution reaches a given confidence level. The proposed noise compensation technique can be used with any EA and can be applied in all evolutionary selections without modifications. There are no limits in the number of objectives that can be handled, and it can be used on single-objective problems as well. While several existing noise compensation approaches assume that all solutions are equally perturbed by noise and/or that the noise characteristics are known at before hand (e.g. [18-20]), the proposed technique does not make any assumption on the noise landscape. For improved efficiency, the technique automatically adapts the number of samplings to the present noise size; solutions subject to much noise are sampled a larger number of times compared to those subject to less noise. The number of samplings are also automatically adjusted according to the importance of the particular solution, as solutions of higher ranks are generally sampled a larger number of times compared to those of lower ranks.

A drawback of the proposed technique is that it involves two user-defined parameters; confidence level and maximum number of samplings. Finding the optimal configuration of these parameters for the problem at hand is a task of trial-and-error. The same drawback of

parameters that must be specified by the user also applies to other noise compensation techniques (e.g. [19][21-22]). Although we have shown that the proposed technique works very well with a standard setting of the parameters (c.f. Section 3), ideally there should be no user-defined parameters at all. Investigating efficient noise compensation techniques that are free from user-defined parameters is an important topic for our future research.

ACKNOWLEDGMENTS

This work has been carried out within the OPTIMIST project which is partially financed by the Knowledge Foundation (KK Stiftelsen), Sweden. The authors gratefully acknowledge the Knowledge Foundation for the provision of research funding and also the industrial partner companies Volvo Aero and Volvo Cars for their support in this study.

8. TEXT STYLES

[1] April, J., Better, M., Glover, F. and Kelly, J. 2004. New advances for marrying simulation and optimization, Proceedings of the 2004 Winter Simulation Conference, Washington, DC, 80-86.

[2] Deb, K. 2004. Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons Ltd.

[3] Jin, Y. and Branke, J. 2005. Evolutionary optimization in uncertain environments – a survey. IEEE Transactions on Evolutionary Computation, 9(3): 303-317.

[4] Bui, L.T., Abbass, H.A. and Essam, D. 2005. Fitness inheritance for noisy evolutionary multi-objective optimization. In Proceedings of Conference on Genetic and Evolutionary Computation, 779-785, Washington DC, USA.

[5] Gosavi, A. 2003. Simulation-based optimization: para-metric optimization techniques and reinforcement learning. Boston, Mass., Kluwer Academic.

[6] Syberfeldt, A., Grimm, H., Ng, A. and John, R.I. 2008. A Parallel Surrogate-Assisted Multi-Objective Evolutionary Algorithm for Computationally Expensive Optimization Problems. In Proceedings of the 2008 IEEE Congress on Evolutionary Computation, Hong Kong, June 1-6.

[7] Ong, Y.S., Nair, P.B., Keane, A.J. and Wong, K.W. 2004. Surrogate-assisted evolutionary optimization frame-works for high-fidelity engineering design problems. Knowledge Incorporation in Evolutionary Computa-tion, Springer Verlag, 307-332.

[8] Ulmer, H., Streichert, F. and Zell, A. 2003. Evolution strategies assisted by gaussian processes with improved pre-selection criterion. In Proceedings of IEEE Congress on Evolutionary Computation, 692-699, Canberra, Australia, December 8-12, 2003.

[9] Streichert, F., Ulmer, H., and Zell, A. 2005. Parallelization of multi-objective evolutionary algorithms using clustering algorithms. In Proceedings of Third Inter-national Conference on Evolutionary Multi-Criterion Optimization (EMO05), 92-107.

[10] Boesel, J., Bowden, R.O., Glover, J.P., Kelly, F. and Westwig, E. 2001. Future of simulation optimization. In Proceedings of Winter Simulation Conference 2001, 1466-1470, Arlington, VA, USA, December 9-12.

[11] Emmerich, M., Giannakoglou, K. and Naujoks, B. 2006. Single- and multi-objective evolutionary optimization assisted by gaussian random field metamodels. IEEE-TEC Special Issue on Evolutionary Computation in the presence of uncertainty, 10(4): 421-439.

[12] Law A.M. and Kelton, D. 200. Simulation Modeling and Analysis, Mc Graw Hill, 3rd edition.

[13] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. 2000. A fast and elitist multi-objective genetic algorithm-NSGA-II. KanGAL Report 2000001, Indian Institute of Technology Kanpur.

[14] Emmerich, M. Beume, N. and Naujoks, B. 2005. An EMO algorithm using the hypervolume measure as selection criterion. In Proceedings of Evolutionary Multi-Criterion Optimization, Springer Berlin-Heidelberg, 62-76.

[15] Jin, Y. 2005. A comprehensive survey of fitness approximation in evolutionary computation. Soft Computing, 9: 3-12, Springer-Verlag Germany.

[16] Mehrotra, K. Mohan, C.K. and Ranka, S. 1996. Elements of Artificial Neural Networks, MIT Press.

[17] Nain P.K.S. and Deb, K. 2005. A multi-objective optimization procedure with successive approximate models. KanGAL report no. 2005002, Indian Institute of Technology, Kanpur, India.

[18] Beyer, H.G. 2000. Evolutionary Algorithms in Noisy Environments: Theoretical Issues and Guidelines for Practice. Computer Methods in Applied Mechanics and Engineering 186(2-4): 239-267.

[19] Hughes, E. Evolutionary Multi-objective Ranking with Uncertainty and Noise. 2001. In Proceedings of First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001), Springer Berlin-Heidelberg, 329-343.

[20] Arnold, D. V. Noisy Optimization with Evolution Strategies. 2002. Kluwer Academic Publishers, Genetic Algorithms and Evolutionary Computation Series.

[21] Babbar, M., Lakshmikantha, A. and Goldberg, D. E. 2003. A modified NSGA-II to solve noisy multiobjective problems. In Proceedings of Genetic and Evolutionary Computation Conference (GECCO2003), Springer Verlag, 21-27.

[22] Büche, D., Stoll, P., Dornberger, R. and Koumoutsakos, P. 2002. An evolutionary algorithm for multi-objective optimization of combustion processes. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 32: 460-473.